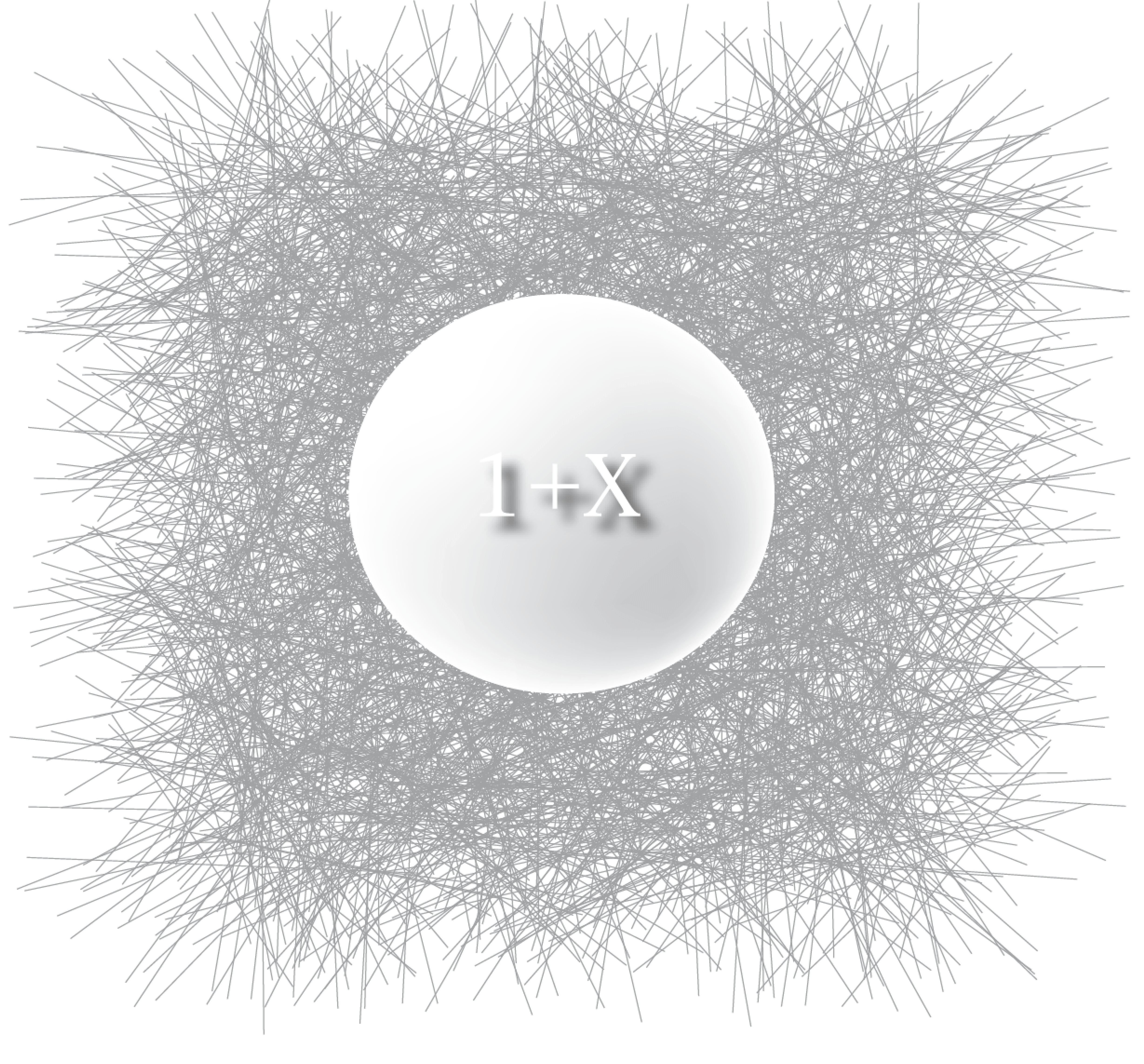
大学计算机基础教育规划教材

Python 程序设计基础

周元哲 编著



清华大学出版社

大学计算机基础教育规划教材

Python 程序设计基础

周元哲　编著

清华大学出版社
北京

内 容 简 介

本书共分14章,内容包括Python编程概述、数据类型和表达式、顺序与选择结构、循环结构、序列与字典、数据结构与算法、函数与模块、面向对象程序设计基础、文件、用户界面设计、绘图、数据库应用、网络编程和异常处理。

本书内容精练、由浅入深,注重学习的连续性和渐进性,章节之间的实例具有关联性。本书适合作为高等院校相关专业Python程序设计的教材或教学参考书,可以供计算机应用开发的各类技术人员参考,亦可作为全国计算机等级考试、软件技术资格与水平考试的培训资料。

图书在版编目(CIP)数据

Python程序设计基础/周元哲编著. —北京:清华大学出版社,2015
大学计算机基础教育规划教材
ISBN 978-7-302-40526-9

Ⅰ. ①P… Ⅱ. ①周… Ⅲ. ①软件工具—程序设计—高等学校—教材 Ⅳ. ①TP311.56

中国版本图书馆CIP数据核字(2015)第136798号

责任编辑: 张 民
封面设计: 傅瑞学
责任校对: 梁 毅
责任印制: 王静怡

出版发行: 清华大学出版社
网 址: http://www.tup.com.cn, http://www.wqbook.com
地 址: 北京清华大学学研大厦A座 **邮 编:** 100084
社 总 机: 010-62770175 **邮 购:** 010-62786544
投稿与读者服务: 010-62776969, c-service@tup.tsinghua.edu.cn
质 量 反 馈: 010-62772015, zhiliang@tup.tsinghua.edu.cn
课 件 下 载: http://www.tup.com.cn, 010-62795954
印 装 者: 保定市中画美凯印刷有限公司
经 销: 全国新华书店
开 本: 185mm×260mm **印 张:** 12.25 **字 数:** 303千字
版 次: 2015年8月第1版 **印 次:** 2015年8月第1次印刷
印 数: 1~2000
定 价: 25.00元

产品编号:064298-01

前　言

Python是一种解释型、面向对象、动态数据类型的高级程序设计语言，在计算机程序设计语言的历史演变中具有划时代的意义。

Python具有简单、现代、类型安全、性能优良等特点，是面向对象程序设计教学的主干语言之一。全书从教学实践角度对 Python进行了全面的阐述，全书共分 14章，内容包括Python编程概述、数据类型和表达式、顺序与选择结构、循环结构、序列与字典、数据结构与算法、函数与模块、面向对象程序设计基础、文件、用户界面设计、绘图、数据库应用、网络编程和异常处理。

在编写本书的整个过程中，作者结合多年从事计算机编程语言的教学经验，在教材内容的选取上力图精简，摒弃陈旧和繁杂的语法规定，只介绍 Python语言的基本语法规定和面向对象的基本特征，培养学生更快地掌握编程思想和编程方法，提高学生的编程应用开发能力。

在本书的编写过程中，西安邮电大学通信与信息工程学院庞胜利、王江舟，计算机科学与技术学院李晓戈、刘伟、张庆生、孟伟君、王小银阅读部分手稿。华东师范大学江红老师等对本书提出了很多宝贵的意见。本书在写作过程中参阅了大量中英文的专著、教材、论文、报告及网上的资料，由于篇幅所限，未能一一列出，在此，一并表示敬意和衷心的感谢。

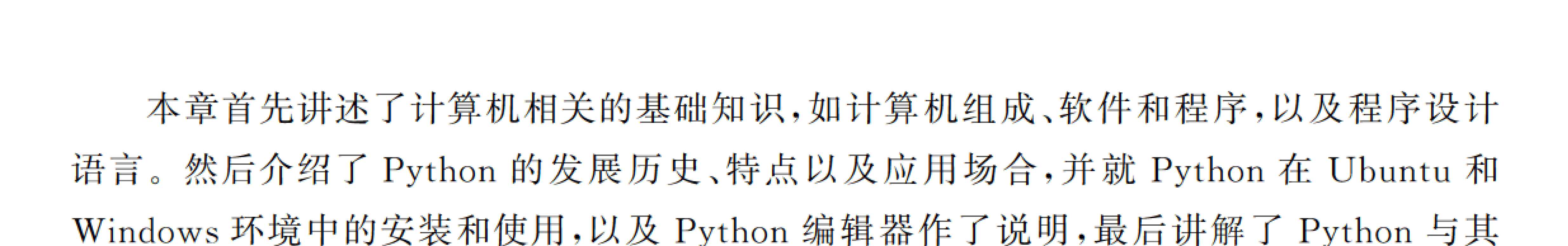
本书内容精练、文字简洁、结构合理、实训题目经典实用、综合性强，明确定位面向初、中级读者，由“入门”起步，侧重“提高”，特别适合作为高等院校相关专业 Python程序设计的教材或教学参考书，也可以供从事计算机应用开发的各类技术人员参考，亦可作为全国计算机等级考试、软件技术资格与水平考试的培训资料。

本书的 Python版本为 2.7.3，所有程序都在 Python自带的 IDE和 notepad++ 编辑器进行调试和运行。由于作者水平有限，时间紧迫，本书难免有疏漏之处，恳请广大读者批评指正。本书作者的电子信箱是 zhouyuanzhe@ 163.com。

作　者
2015 年 3 月

目　录

第1章 Python编程概述



本章首先讲述了计算机相关的基础知识，如计算机组成、软件和程序，以及程序设计语言。然后介绍了 Python 的发展历史、特点以及应用场合，并就 Python 在 Ubuntu 和 Windows 环境中的安装和使用，以及 Python 编辑器作了说明，最后讲解了 Python 与其他语言之间的关系。

1.1 计算机基础知识

本节从计算机组成、软件和程序、程序设计语言等方面进行介绍。

1.1.1 计算机组成

计算机是由集成电路组成的电子设备，它由两部分组成：硬件系统和软件系统。所有可视的设备和外围设备都属于硬件系统。1944 年，美籍匈牙利数学家冯·诺依曼提出了计算机基本结构和工作方式的设想，为计算机的诞生和发展提供了理论基础。时至今日，尽管计算机软硬件技术飞速发展，但计算机本身的体系结构并没有明显的突破，当今的计算机的体系结构仍属于冯·诺依曼架构。

冯·诺依曼提出的理论要点有以下两点：

(1) 计算机硬件设备由存储器、运算器、控制器、输入设备和输出设备五部分组成。其中，运算器和控制器组成中央处理器单元(CPU，Center Process Unit)。中央处理单元用于执行指令，如算术操作、从别的设备写入或读出数据。存储器分为内存和外存。CPU 从内存中读取所需要的数据，进行处理。内存中存储的数据是临时的，当程序退出或者计算机关机时，数据将会丢失。如果需要永久存储数据，需要用到外存，如硬盘、闪存等设备。键盘、鼠标等输入设备用于接受用户输入数据和指令，显示器通常作为输出设备。

(2) 存储程序思想——把计算过程描述为由许多命令按一定顺序组成的程序，然后把程序和数据一起输入计算机，计算机对已存入的程序和数据处理后，输出结果。图 1.1 为计算机体系结构图。

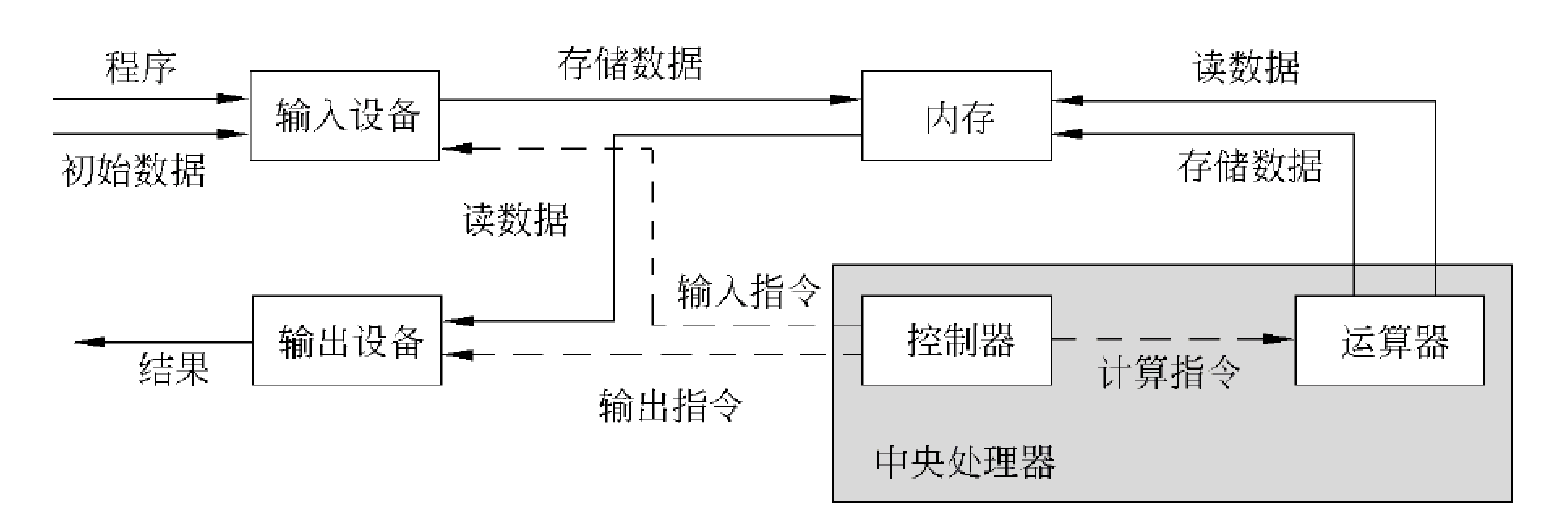


图 1.1　计算机体系结构图

1.1.2　软件和程序

相对于硬件系统而言，软件系统是由一些“不可视”的部分组成的，它是一系列按照特定顺序组织的计算机数据和指令的集合，如程序、数据、音频、视频等。实际上，不论指令还是数据都以二进制编码形式存在计算机中。在二进制系统中只有两个数(0 和 1)，这是因为计算机硬件组成的物理器件具有两种稳定状态，如门电路的导通与截止、电压的高与低，恰好对应表示 1 和 0 两个符号。

计算机软件一般分为系统软件和应用软件两大类。系统软件为计算机用户提供最基本的功能，一般是操作系统和通用平台，如 UNIX，Windows，Linux 和 Android 等，帮助用户管理计算机的硬件。应用软件则是为了特定目的而设计的软件，不同的应用软件根据用户和所服务的领域提供不同的功能，如 Office，Photoshop 和游戏软件等。

一般认为，软件包括以下一些内容：

(1) 运行时，能够提供所要求功能和性能的指令或计算机程序集合。

(2) 程序能够满意地处理信息的数据结构。

(3) 描述程序功能需求、程序如何操作和使用所要求的文档。

软件和程序是两个概念，对于初学者往往会混淆。其实，这发生在软件发展历史的第一阶段(20 世纪 50 年代初期至 60 年代中期)，由于软件的生产个体化，规模较小，功能单一，软件只有程序而无文档，形成了“软件等于程序”的错误观念。程序是为实现特定目标或解决特定问题而用计算机语言编写的命令序列的集合，通过使用与自然语言具有相似的语法和语义的程序设计语言编写源代码，利用特定的工具将其翻译成 CPU 所能执行的指令，完成特定的目的。

1.1.3　程序设计语言

程序设计语言的发展经过了以下几个阶段。

1. 第一代程序设计语言

机器语言是用二进制代码表示的计算机能直接识别和执行的一种机器指令的集合，指令是由 0 和 1 组成的一串代码，通过线路变成电信号，让计算机执行各种不同的操作。机器语言具有直接执行特点。编程人员需要熟记所用计算机的全部由若干个 0 和 1 组成

的指令代码和代码的含义，机器语言具有难读、难编、难记和易出错的缺点。

2. 第二代程序设计语言

为了克服机器语言的缺点，人们用与代码指令实际含义相近的英文缩写词、字母和数字等符号来取代指令代码(如用 ADD 表示运算符号“+”的机器代码)，采用助记符号编写程序，于是就产生了汇编语言。汇编语言比用机器语言的二进制代码编程要方便些，在一定程度上简化了编程过程。

汇编语言又称为符号语言，不能直接被机器识别，要使用一种程序将汇编语言翻译成机器语言。当汇编语言产生面向硬件操作控制信息的指令时，使用起来依旧烦琐，程序无结构，通用性也差。但是，使用汇编语言来编制系统软件和过程控制软件，其目标程序占用内存空间少，运行速度快，有着高级语言不可替代的用途。

3. 第三代程序设计语言

由于机器语言、汇编语言依赖于硬件体系，要求使用者必须对硬件结构及其工作原理都十分熟悉，因此人们又发明了与人类自然语言相接近且能为计算机所接受的规则明确、通用易学的计算机语言，其语法和结构具有类似文字的表现形式。1954 年，第一个面向科学计算的高级计算机语言——FORTRAN 语言被正式推广使用，FORTRAN 语言是 Formula Translation 的缩写，意为“公式翻译”，是数值计算领域使用的主要语言。1972 年，作为程序语言的里程碑，C 语言诞生了，它不但具有高级语言的特点，又具有汇编语言的特点，逐渐成为教学科研和软件开发的主要语言。

4. 第四代程序设计语言

面向对象程序设计语言、脚本语言、人工智能语言等通常被认为是第四代程序设计语言。SIMULA67 是第一个面向对象程序设计语言，特别是 1995 年 5 月由 Sun 公司推出的 Java 程序设计语言，可以撰写跨平台应用软件。第四代程序设计语言提供了功能强大的非过程化问题定义手段，用户只须告知系统做什么，而无须说明怎么做，因此可大大提高软件生产效率。Python 就属于第四代程序设计语言。

1.2 Python 的发展历史

Python 是一种解释型、面向对象、动态数据类型的高级程序设计语言，被列入 LAMP (Linux, Apache, MySQL 以及 Python/Perl/PHP)。Python 由 Guido van Rossum 于 1989 年年底发明，第一个公开发行版发行于 1991 年。像 Perl 语言一样，Python 源代码同样遵循 GPL(General Public License，翻译为 GNU 通用公共许可证)协议。

Python 2.0 于 2000 年 10 月 16 日发布，实现垃圾回收，并支持 Unicode。Python 的 3.0 版本于 2008 年 12 月 3 日发布，常被称为 Python 3000，或简称 Py3k，相对于 Python 的早期版本，作了较大的升级。由于未考虑向下相容，导致早期 Python 版本设计的程序无法在 Python 3.0 上正常执行。为此，Python 2.6 和 2.7 作为一个过渡版本，基本使用

了 Python 2.x 的语法和库，同时考虑了向 Python 3.0 的迁移，允许使用部分 Python 3.0 的语法与函数。

本书的所有程序都在 Python 2.7.3 版本的开发环境中进行调试和运行。

1.3 Python 的特点

Python 是一种简单易学、功能强大的编程语言，它有高效率的高层数据结构，可以简单而有效地实现面向对象编程。Python 具有如下一些特点：

1. 简单易学

Python 作为代表简单主义思想的语言，其语法简洁而清晰，结构简单，可以快速上手，易于学习，Python 在学习过程中不用计较程序语言中在形式上的诸多细节和规则，便于专注程序本身的逻辑和算法，探究程序执行的过程。

2. 免费开源

Python 是 FLOSS(自由/开放源码软件)之一，可以自由地发布这个软件的副本，阅读它的源代码，对它做改动，并将它用于新的自由软件中。

3. 解释型语言

计算机并不能直接接收和执行用高级语言编写的源程序，源程序在输入计算机时，通过“翻译程序”翻译成机器语言形式的目标程序，计算机才能识别和执行。这种“翻译”通常有两种方式：一种是编译执行；另一种是解释执行。

编译执行是指源程序代码先由编译器编译成可执行的机器码，然后再执行；解释执行是指源代码程序被解释器直接读取执行。编译执行和解释执行各有优缺点，编译执行可一次性将高级语言源程序编译成二进制的可执行指令，通常执行效率高。而解释执行是由该语言(如 HTML)运行环境(如浏览器)读取一条该语言的源程序，然后转变成二进制指令交给计算机执行，通常可以灵活地跨平台。C，C++ 等采用编译执行方式，Python 作为解释型语言，与 Java 语言类似，不需要编译成二进制代码，通过解释器把源代码转换成称为字节码的中间形式，由虚拟机负责在不同的计算机运行，因此，Python 程序便于移植，可在众多平台运行，如 Linux，Windows，FreeBSD，Macintosh，Solaris，OS/2，Amiga，AROS，AS/400，BeOS，OS/390，z/OS，Palm OS，QNX，VMS，Psion，Acom RISC OS，VxWorks，PlayStation，Sharp Zaurus 和 Windows CE 等。

4. 面向对象

Python 是完全面向对象的语言。函数、模块、数字、字符串都是对象，并且完全支持继承、重载、派生和多重继承。Python 语言编写程序无须考虑硬件和内存等底层细节。

5. 丰富的库

Python 称为胶水语言，能够轻松地与其他语言(特别是 C 或 C++)联结在一起，其具有丰富的 API 和标准库，可以完成各种工作，包括正则表达式、文档生成、单元测试、线程、数据库、网页浏览器、CGI、FTP、电子邮件、XML、XML-RPC、HTML、WAV 文件、密码系统和 GUI 等。

1.4 Python 的应用场合

Python 功能强大，主要应用于以下场合：

(1) GUI 软件开发

Python 具有 wxPython 和 PyQT 等工具，使得 Python 可以快速开发出 GUI，并且不做任何改变就可以运行在 Windows，Xwindows 和 MacOS 等平台。

(2) 网络应用开发

Python 提供了标准 Internet 模块，可以广泛应用到各种网络任务中，无论在服务器端还是在客户端，另外，网站编程第三方工具：HTMLGen，mod_python，Django，TurboGears 和 Zop 能够帮助 Python 快速构建功能完善和高质量的网站。

(3) 游戏开发

Pygame 是建立在 SDL(Simple DirectMedia Layer)基础上的软件包，提供了简单的方式控制媒体信息(如图像和声音等)，专为电子游戏设计使用。Pygame 下载网址为 www.pygame.org，如图 1.2 所示。

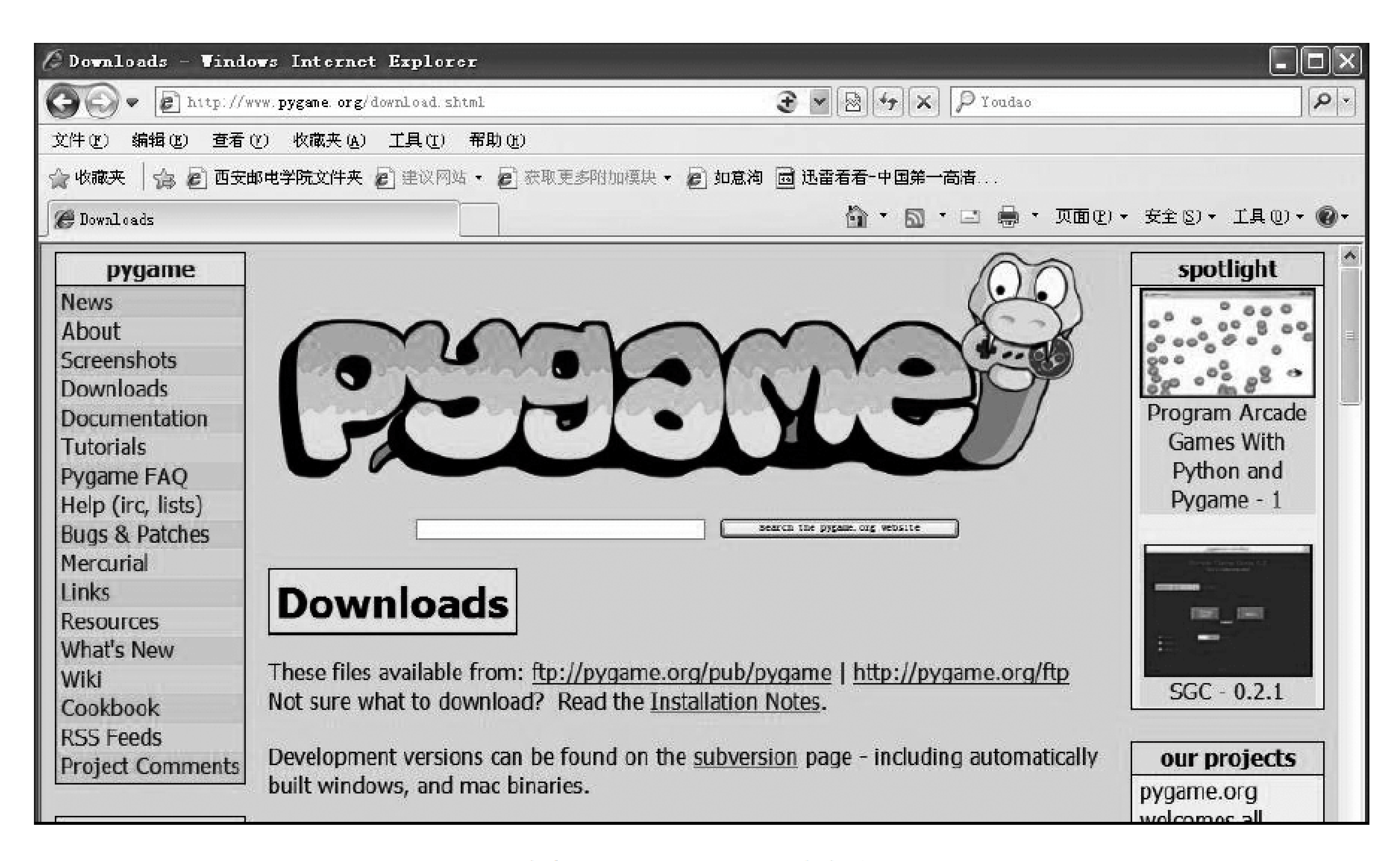


图 1.2 Pygame 网址

本书 Python 版本为 2.7.3，故下载 pygame-1.9.1.win32-py2.7.msi 3.1MB 文件。

(4) 科学计算

随着 NumPy，SciPy，Matplotlib，Enthought librarys 等众多程序库的开发，Python 越

来越适合于做科学计算、绘制高质量的 2D 和 3D 图像。和科学计算领域最流行的商业软件 Matlab 相比，Python 是一门通用的程序设计语言，比 Matlab 所采用的脚本语言的应用范围更广泛，有更多的程序库的支持。

(5) Web 与移动设备应用开发

web2py 是一种免费的开源的 Web 开发框架，帮助开发者分别设计、实施和测试 MVC(模型(Model)、视图(View)、控制器(Controller))模型。web2py 下载网址为：www.web2py.com，如图 1.3 所示。



图 1.3　web2py 网址

(6) 数据库开发

Python 支持所有主流数据库，如 Oracle，Sybase，MySQL，PostgreSQL 和 Informix 等，并通过标准的数据库 API 接口将关系数据库映射到 Python 类实现面向对象数据库系统。

(7) 系统编程

Python 对操作系统服务设置的内置接口，使其成为编写可移植的维护操作系统的管理工具和部件，Python 程序可以搜索文件和目录树，可以运行其他程序，用进程或线程进行并行处理等。

1.5　Python 解释器

1.5.1　在 Ubuntu 下安装 Python

Ubuntu(乌班图)是一个以桌面应用为主的 Linux 操作系统，基于 Debian 发行版和 GNOME 桌面环境，与 Debian 的不同在于它每 6 个月会发布一个新版本。Ubuntu 的目标在于为用户提供最新的，同时又相当稳定的自由软件构建的操作系统。

Ubuntu 下内置 Python,如图 1.4 所示。

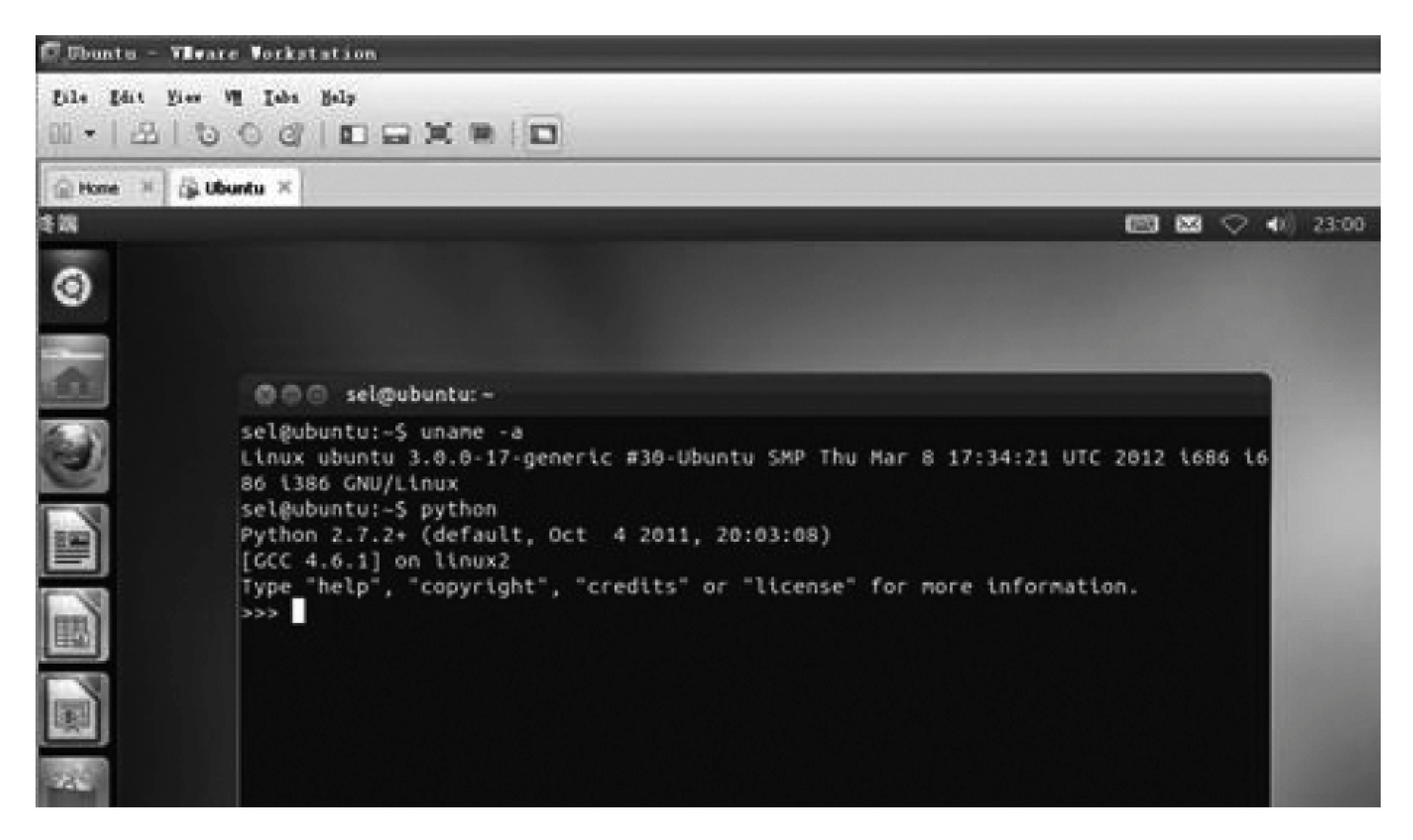


图 1.4　ubuntu 下内置 Python

1.5.2　在 Windows 下安装 Python

Windows 下安装 Python,一般具有如下步骤:

步骤一:下载 Python2.7.3 安装包进行安装。

在浏览器中输入 http://www.Python.org,在下载页 https://www.python.org/download/releases/2.7.3/中找到 Windows x86 MSI Installer(2.7.3) (sig)进行下载,如图 1.5 所示。

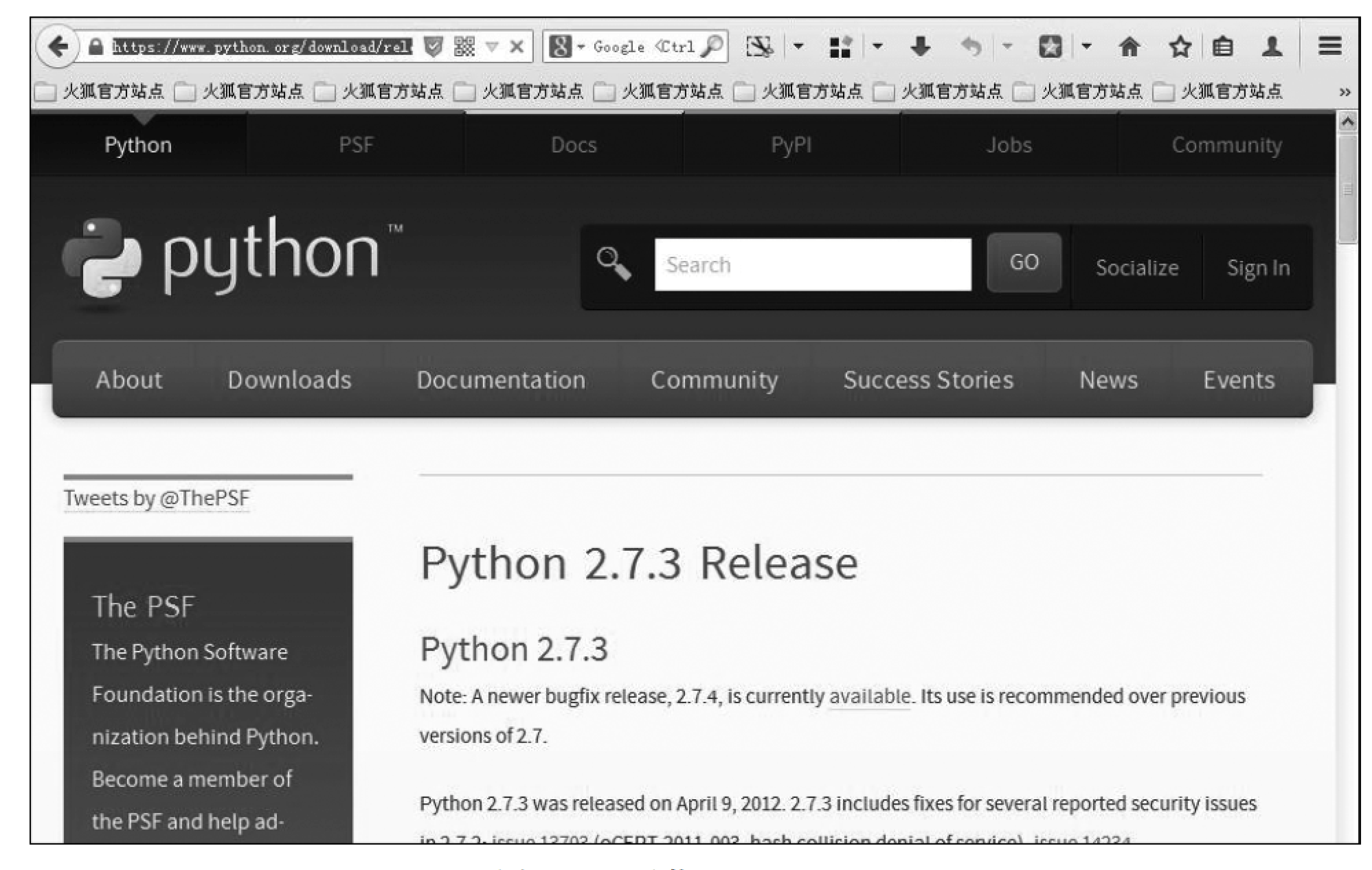


图 1.5　下载 Python2.7.3

步骤二：在 Windows 环境变量中添加 Python。

将 Python 的安装目录添加到 Windows 下的 path 变量中，如图 1.6 所示。



图 1.6　设置环境变量

步骤三：测试 Python 安装是否成功。

在 Windows 下使用 cmd 打开命令行，输入 Python 命令，图 1.7 表示安装成功。

```
C:\WINDOWS\system32\cmd.exe - python
Microsoft Windows XP [版本 5.1.2600]
(C) 版权所有 1985-2001 Microsoft Corp.

C:\Documents and Settings\Administrator>python
Python 2.7.3 (default, Apr 10 2012, 23:31:26) [MSC v.1500 32 bit (Intel)] on win
32
Type "help", "copyright", "credits" or "license" for more information.
>>> print 'hello'
hello
>>>
```

图 1.7　测试 Python 安装是否成功

1.6　Python 编辑器

Python 编辑器众多，有 Python 自带的 IDE 编辑器、notepad++、Eclipse＋pydev、Ulipad 以及 Vim 和 emacs 等。下面依次进行介绍。

1.6.1　IDLE

IDLE 作为 Python 安装后内置的集成开发工具，包括能够利用颜色突出显示语法的编辑器，调试工具，Python Shell 以及完整的 Python 3 在线文档集。

Python 的 IDLE 具有命令行和图形用户界面两种方式，采用命令行交互式执行 Python 语句，方便快捷。但必须逐条输入语句，不能重复执行，适合测试少量的 Python 代码，不适合复杂的程序设计。

Windows 下安装的 Python 文件如图 1.8 所示。



图 1.8　IDLE

1.6.2　Notepad++

Notepad++ 是在微软视窗环境之下的免费代码编辑器，功能完全取代记事本，内置支持多达 27 种语法高亮度显示(包括各种常见的源代码、脚本，能够很好地支持.nfo 文件查看)，还支持自定义语言；可自动检测文件类型，根据关键字显示节点，节点可自由折叠或打开，还可显示缩进引导线，代码显示得很有层次感；可打开双窗口，在分窗口中又可打开多个子窗口等功能。

Notepad++ 编辑器的安装经过如下步骤：

步骤一：安装 Python2.7.2 版本，默认安装路径为 C:\Python27。

步骤二：安装 Notepad++ 。下载网址为：

http://xiazai.sogou.com/detail/34/16/-2673472578065113427.html?w=1927，下载 npp.6.6.9.Installer 文件后，双击安装，如图 1.9 所示。



图 1.9　Notepad++ 安装界面

Notepad++ 运行界面如图 1.10 所示。

步骤三：配置 Notepad++ 运行环境，其运行环境配置具体如下所示：

(1) 运行 Notepad++ (按下 F5 键)，输入如下语句到输入框。

cmd/k python " $(FULL_CURRENT_PATH)" & PAUSE & EXIT 单击“保存…”按钮，如图 1.11 所示。

(2) 在 Shortcut 窗口设置运行命令的快捷键，如图 1.12 所示。

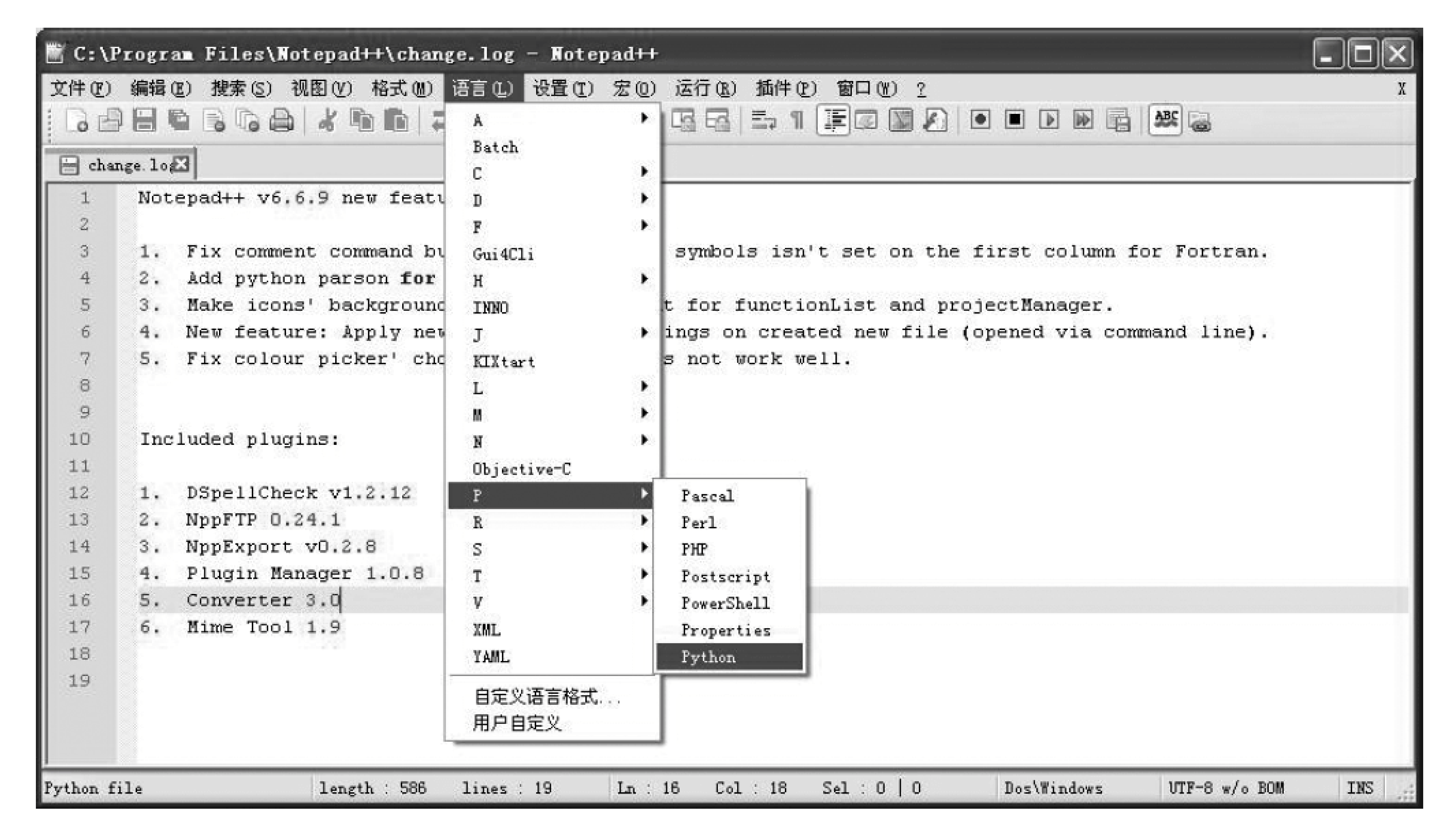


图 1.10 Notepad++运行界面



图 1.11 设置 Notepad++ 宏定义

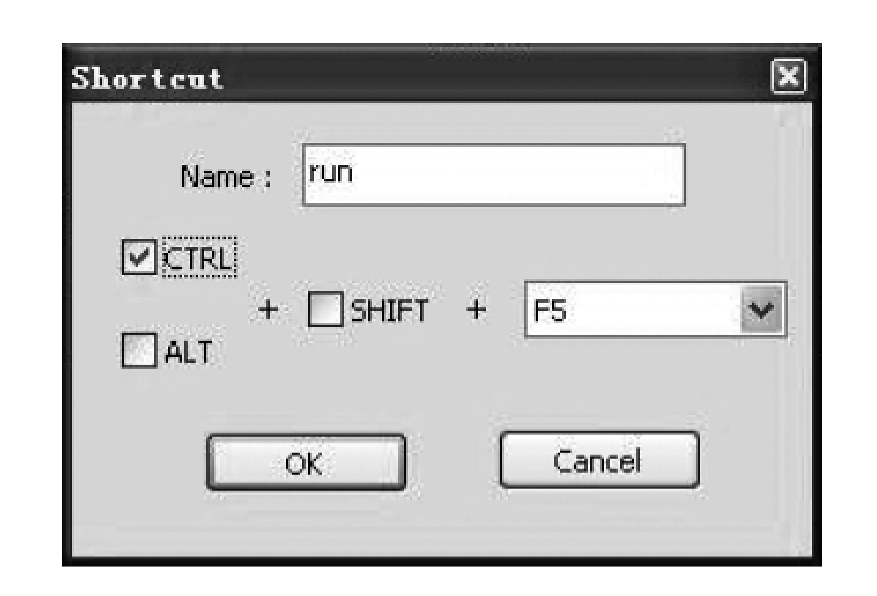


图 1.12 设计 Notepad++运行命令的快捷键

其中,Name 可以随便输入(例如:run),快捷键也可由用户自行选择,但必须不能跟系统中已有的快捷键设置冲突(例如 Ctrl+F5 键)。设置完以后,单击 OK 按钮保存此命令。

1.6.3 Ulipad

Ulipad 的前身是 NewEdi,轻便小巧而功能强大,非常适合初学者。Ulipad 编辑器需要 wxPython 和 comtypes 两个软件支持方能实现 Python 的编程。

Ulipad 编辑器的安装经过如下步骤:

步骤一:安装 Python2.7.2 版本,默认安装路径为 C:\Python27。

步骤二:安装 wxPython,在第 10 章详细介绍。

步骤三:安装 comtypes,comtypes 的下载网址:

http://sourceforge.net/projects/comtypes/files/,下载 32 位的 comtypes-0.6.2.win32 文件,双击安装,如图 1.13 所示。

wxPyhton 和 Comtypes 安装的目录都是 Python 安装的默认路径 C:\Python27。

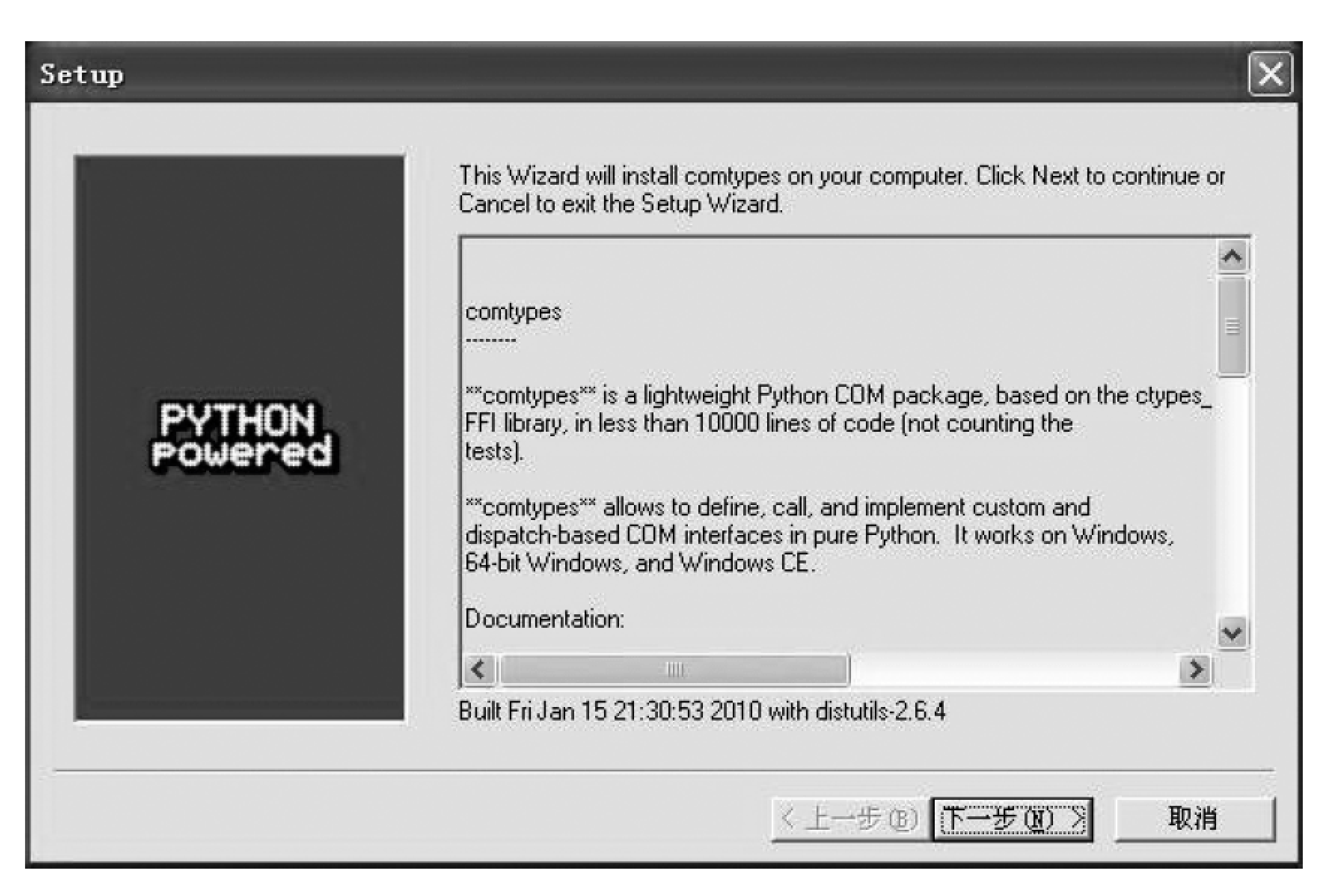


图 1.13 comtypes 安装截图

步骤四：安装 Ulipad。

由于 Python 是 32 位的 2.7.3 版本，需下载 ulipad.4.1.py27 文件，下载网址为 http://code.google.com/p/ulipad/downloads/list，安装默认路径 C:\Program Files\UliPad，如图 1.14 所示。



图 1.14 Ulipad 安装截图

步骤五：配置 Ulipad。

安装 Ulipad 后，在 Ulipad 的安装目录 C:\Program Files\UliPad 找到 UliPad 程序文件，将其重新命名为 UliPad -n，双击运行，如图 1.15 所示。

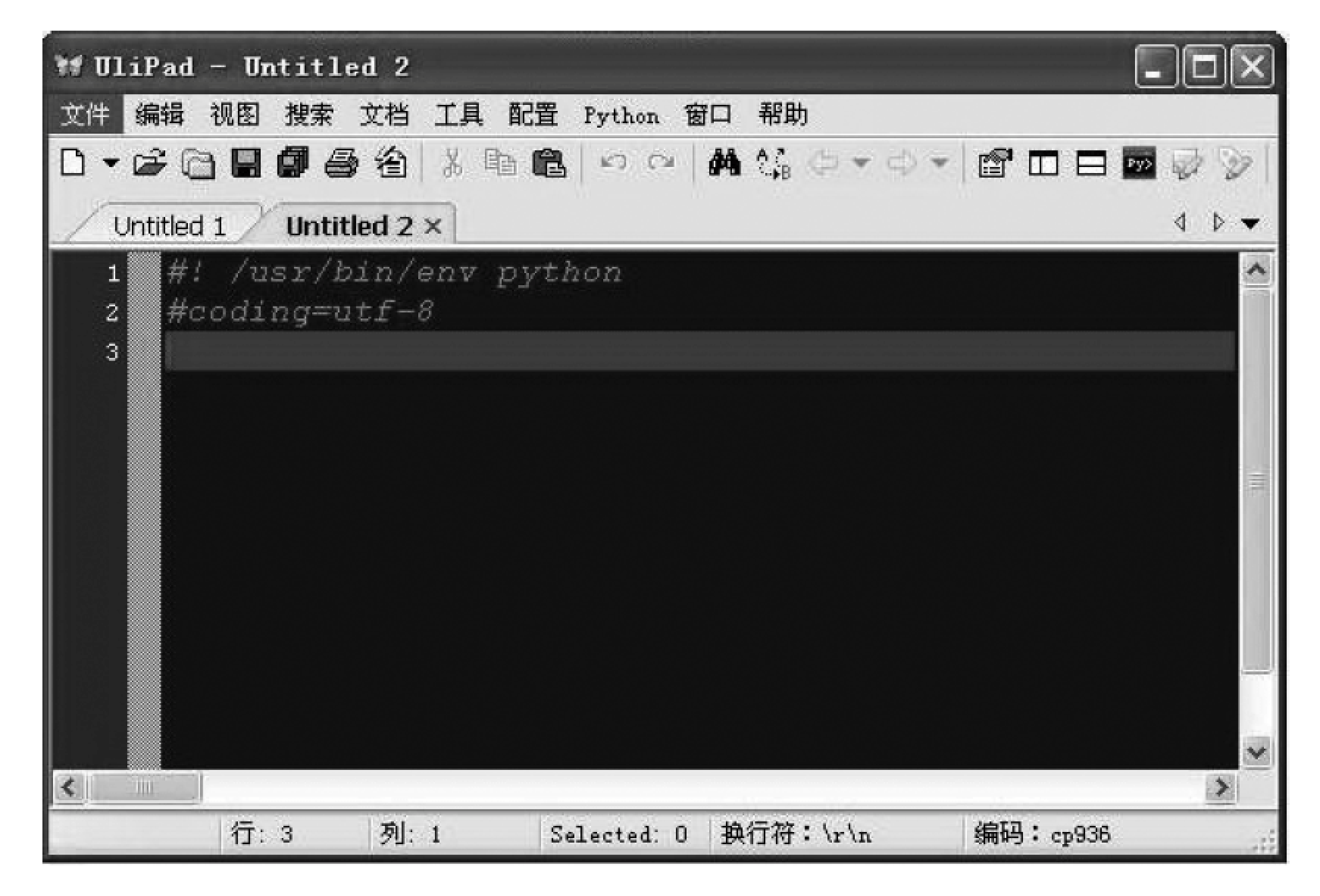


图 1.15　Ulipad 运行界面

1.6.4　Eclipse＋PyDev

Eclipse 和 PyDev 插件结合可以为 Python 开发提供完全集成的开发环境和构建/部署工具。

Eclipse＋PyDev 编辑器的安装经过如下步骤：

步骤 1：安装 Python2.7.3 版本，默认安装路径为 C:\python27。

步骤 2：下载安装 32 位的 JDK6 Java，下载 32 位的 Eclipse，Eclipse 官网下载：http://www.eclipse.org/。

步骤 3：下载安装 PyDev 插件。

启用 Eclipse，在 Help 菜单中，选择安装新软件，如图 1.16 所示。



图 1.16　步骤 1

单击"添加"按钮，名称输入 PyDev，位置输入 http://pydev.org/updates，单击"确定"按钮，如图 1.17 所示。

图 1.17 步骤 2

由于 Python 是 2.7.3 版本，需要将"只显示可用软件的最新版本"选项去掉，以便可以选择与其相同的版本。安装 PyDev 下的 PyDev for Eclipse 版本为 2.7.3，如图 1.18 所示，安装完成后，重启 Eclipse。

图 1.18 步骤 3

步骤 4：配置 PyDev。

选择 Eclipse 的"窗口"→"首选项"→Interpreter→ Python，New 一个 Python 解释器，填上解释器名字和路径，路径选相应的 python.exe。

步骤 5：测试安装环境。

1.6.5 Vim 和 emacs

Vim 是从 vi 发展出来的一个文本编辑器，具有代码补完、编译及错误跳转等功能，

Vim 和 emacs 作为 Linux 的编辑软件，并列成为类 UNIX 系统中程序员广泛使用的软件。

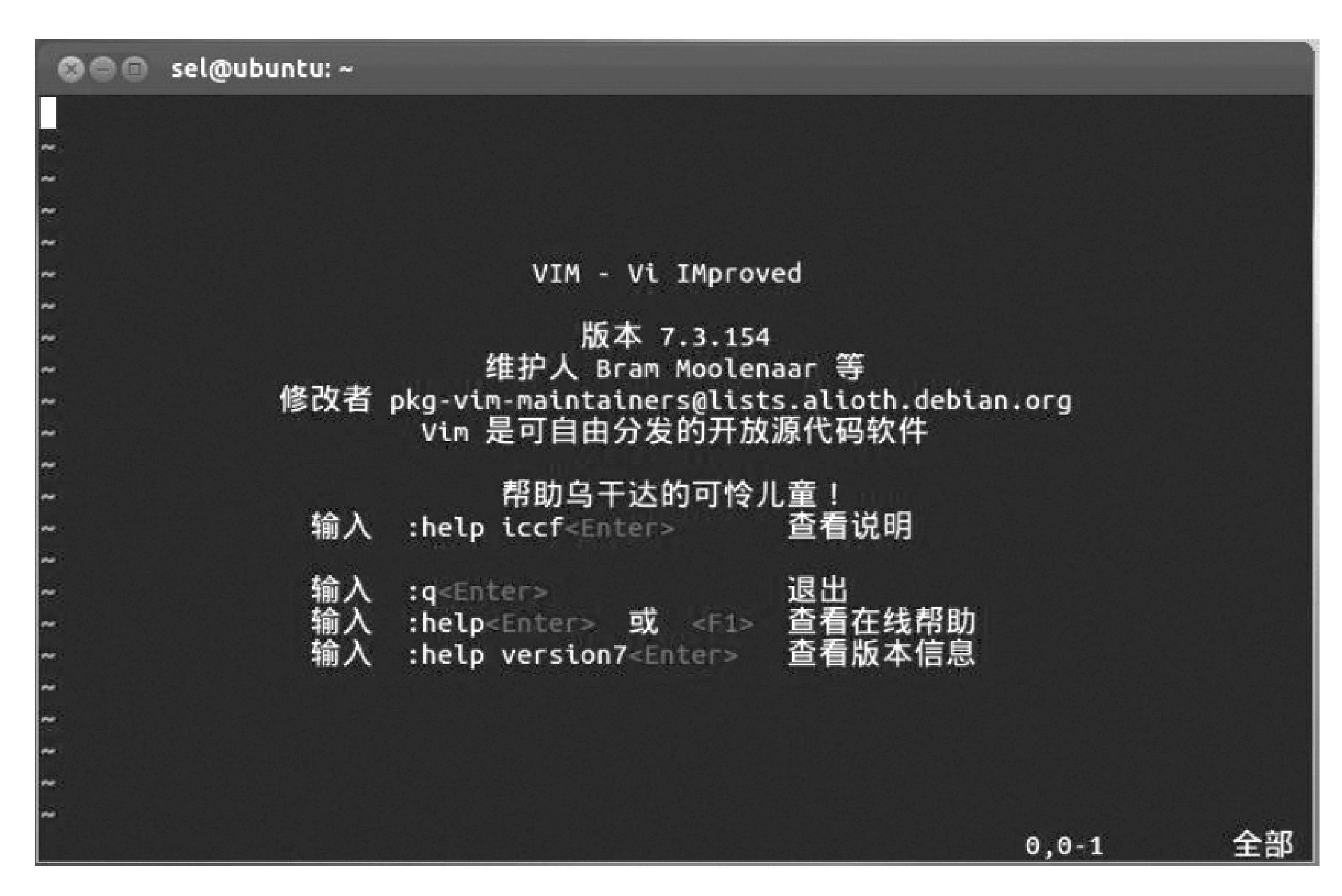


图 1.19 ulipad 安装截图

1.7 Python 与其他语言关系

下面依次介绍 Python 与 C 和 Java 的关系。

Python 语言与 C 语言的区别如下：

(1) Python 是动态编译语言，变量的使用只须赋值；c 是静态编辑语言，变量的使用必须先定义，并显示说明其类型方法能使用。

(2) Python 有列表、元组、字典等数据结构，而 C 语言没有。

(3) Python 是弱类型语言，C 语言是强类型语言。

Python 语言与 Java 语言的关系如下：

(1) Python 与 Java 都支持面向对象编程。

(2) Python 比 Java 要简单，非常适合构造快速原型。

(3) Python 和 Java 都适合程序员协同开发大型项目。

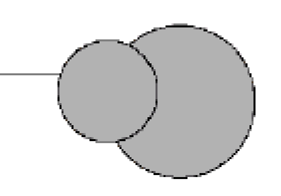
总之，Python 具有如下功能：

- 比 TCL 强大，支持“大规模编程”，适于开发大型系统；
- 比 Perl 语法简洁，更具可读性、更易于维护，有助于减少 Bug；
- 比 Java 更简单、更易于使用；
- 比 C++ 更简单、更易于使用；
- 比 VB 更强大也更具备跨平台特性；
- 比 Ruby 更成熟、语法更具可读性。

1.8　习题

1. 冯·诺依曼理论是什么？
2. 软件和程序是否一样？
3. 程序设计语言经过了哪些阶段？
4. 简述 Python 的功能和特点。
5. 安装 Notepad++，学习如何使用。
6. 安装 Ulipad，学习如何使用。
7. 安装 Eclipse＋PyDev，学习如何使用。
8. Python 相比其他程序设计语言有什么特点？

第2章 数据类型和表达式



本章主要介绍 Python 的数据类型、变量和常量、算术运算符、关系运算符、逻辑运算符、身份运算符等和表达式的运算，最后讲解了一些常用系统函数以及 Python 的字符。

2.1 数据类型

计算机能处理数值、文本、图形、音频、视频、网页等各种数据，不同的数据需要定义不同的数据类型。数据类型是指根据数据描述信息的含义，将数据分为不同的种类。例如，年龄为 25，用整数来表示；成绩 78.5 用单精度来表示；姓名如“比尔 · 盖茨”，用字符串来表示等。

Python 的数据类型分为数值、字符串、布尔型、空值等。

2.1.1 数值

数值分为整数类型、浮点数和复数。

1. 整数

十进制整数，如 0，−1，9，123。

十六进制整数，需要 16 个数字：0，1，2，3，4，5，6，7，8，9，a，b，c，d，e，f 表示整数，为了告诉计算机这是一个十六进制数，必须以 0x 开头，如 0x10，0xfa，0xabcdef。

八进制整数，需要 8 个数字：0，1，2，3，4，5，6，7 表示整数，为了告诉计算机这是一个八进制数，必须以 0 开头，如 035，011。

二进制整数，需要 2 个数字：0，1 表示整数，为了告诉计算机这是一个二进制数，必须以 0b 开头，如 0b101，0b100。

2. 浮点数

浮点数又称小数，如 15.0，0.37，−11.2，1.2e2，314.15e-2。

3. 复数

复数是由实部和虚部构成的数，如 3+4j，0.1−0.5j。

2.1.2 布尔型

在 Python 中,布尔型只有 true 和 false 两个取值。以下数值会被认为是 false:为 0 的数字(包括 0,0.0);空字符串(包括'',"");表示空值的 None;空集合(包括(),[],{})。其他值都认为是 true。

2.1.3 字符串

用单引号、双引号或三引号括起来的符号系列称为字符串,如'Hello World',"Python is very easy"等。用单引号或双引号括起来没有任何区别,只是一个字符串用什么引号开头,就必须用什么引号结尾。单引号与双引号只能创建单行字符串。

为了在字符串数据里使用双引号,出现了三引号。三引号也可以创建多行字符串。

【例 2-1】 字符串举例。

```
>>> print '''we say "hello" to python'''
we say "hello" to python
>>> print ''' hello!,
everyone! '''
  hello!,
everyone!
```

2.1.4 空值

空值是 Python 里一个特殊的值,用 None 表示。

2.2 变量与常量

2.2.1 标识符

标识符是用来标识变量的名字。命名标识符必须遵循以下规则:

(1) 变量名可以由字母、数字和下划线组成。

(2) 变量名的第一个字符必须是字母或者下划线"_",但不能以数字开头。

(3) 变量名不能和关键字同名。

(4) 变量名区分大小写,myname 和 myName 不是同一个变量。

(5) 以双下划线开头的标识符是有特殊意义,是 Python 采用特殊方法的专用标识,如__init__()代表类的构造函数。

例如,a123、XYZ、变量名、sinx 等都符合变量的命名规则,是正确的。而下面的变量命名不符合变量命名规则,因此都是错误的。

```
3xy         '变量名必须以字母开头,不能以数字开头。
ab%c        '变量名可以由字母、数字和下划线组成,不能含有非法字符"%"。
Wang ping   '变量名不能包含空格。
```

注意：变量的命名方法采用匈牙利命名法。匈牙利命名法采用小写前缀与有特定描述意义的名字相结合的方式来为变量命名。

2.2.2 变量

Python 是一种动态类型语言，即变量不需要显式声明数据类型。Python 认为任何数据都是“对象”，变量用来指向对象，变量赋值就是把对象和变量关联起来，变量名是对对象数据的引用，多个变量可以指向同一个对象。每次变量重新赋值，并没有改变对象的值，只是新创建了一个新对象，并用变量指向它，从变量到对象的连接称为引用，如图 2.1 所示。

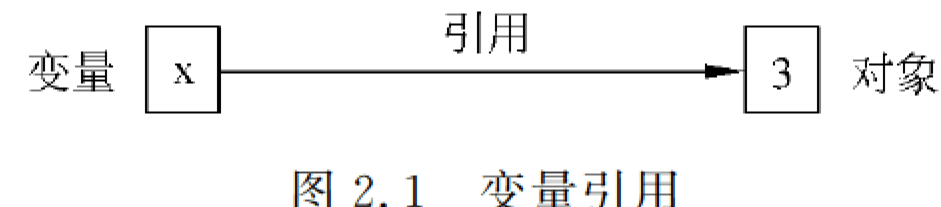


图 2.1 变量引用

Python 中变量的声明是通过变量的赋值操作实现，具有如下特点：

- 变量和对象一样不需要声明；
- 变量在第一次赋值时创建。

【例 2-2】 变量举例。

```
>>>a='ABC'
```

【解析】 Python 解释器做了如下两件事情：

(1) 在内存中创建了一个'ABC'的字符串；

(2) 在内存中创建了一个名为 a 的变量，并把它指向'ABC'。

【例 2-3】 变量声明举例。

```
>>>a=1
print a
a='hello'
print a
a=True
print a
```

【解析】 变量 a 先后成为了整数、字符串、bool 类型。

2.2.3 常量

常量就是不能变的变量，例如圆周率 π 就是一个常量。在 Python 中，通常用全部大写的变量名表示常量，例如：

```
PI=3.14159265359
```

2.3 运算符

运算符是表示实现某种运算的符号。Python 具有丰富的运算符，可分为算术运算符、关系运算符、逻辑运算符、身份运算符、位运算符等。

2.3.1　算术运算符

算术运算符如表 2.1 所示。

表 2.1　算术运算符

运算符	描　　述	实　　例
+	两个对象相加	10+3 得到 13，'a'+'b'得到 'ab'
-	得到负数或是一个数减去另一个数	3-10 得到 -7
*	两个数相乘或是返回一个被重复若干次的字符串	3 * 3 * 3 得到 27 'la' * 3 得到 'lalala'
/	除	10/3 得到 2.333 333 333 333 33
//	取整除，用于得到商的整数部分	9//2 输出结果 4，9.0//2.0 输出结果 4.0
%	取模运算，返回除法的余数	10%3 得到 1
**	幂运算	3**2 得到 9

【例 2-4】 计算表达式 5+10 % 10//9/3+2**2 的值。

【解析】 运算步骤如下：

步骤 1：找出所有的运算符：+、%、//、/、+、**。

步骤 2：将运算符的优先级进行排序：**、/、//、%、+。

步骤 3：加入必要的括号改变表达式运算的先后次序，如下所示：

5+(10%(10//(9/3)))+(2**2)

步骤 4：依次进行运算 2**2=4，9/3=3，…，结果为 10。

注意： 为了验证 Python 表达式运算结果，可以使用 print 查看结果。

```
print  5+(10%(10//(9/3)))+(2**2)
```

2.3.2　关系运算符

关系运算符又称比较运算符，是双目运算符，作用是将两个操作数的大小进行比较，比较的结果是一个布尔值，即 true（真）或 false（假）。操作数可以是数值型或字符型。表 2.2 列出了 Python 中的关系运算符。

表 2.2　关系运算符

运算符	描　　述	实　　例
==	等于	"ABCDE"=="ABR" 返回 false
>	大于	"ABCDE">"ABR" 返回 false
>=	大于或等于	"bc">="大小"返回 false
<	小于	23<3 返回 false
<=	小于或等于	"23"<="3" 返回 true
!=	不等于	"abc"!="ABC" 返回 true

关系运算符在进行比较时，需注意以下规则：

(1) 两个操作数是数值型，则按大小进行比较。

(2) 两个操作数是字符型，则按字符的 ASCII 码值从左到右逐一进行比较，即首先比较两个字符串中的第 1 个字符，ASCII 码值大的字符串为大，如果第一个字符相同，则比较第 2 个字符，依此类推，直到出现不同的字符时为止。

2.3.3 逻辑运算符

逻辑运算符如表 2.3 所示。除 Not 是单目运算符外，其余都是双目运算符，逻辑运算结果是布尔值 true 或 false。

表 2.3 逻辑运算符

运算符	含义	描　　述	实　例	结果
Not	取反	当操作数为假时，结果为真，当操作数为真时，结果为假	Not F Not T	T F
And	与	当两个操作数均为真时，结果才为真；否则为假	T And T F And F T And F F And T	T F F F
Or	或	当两个操作数至少有一个为真时，结果为真；否则为假	T Or T F Or F T Or F F Or T	T F T T

2.3.4 身份运算符

身份运算符用于比较两个对象的存储单元，如表 2.4 所示。

表 2.4 身份运算符

运算符	描　　述	实　　例
is	is 是判断两个标识符是不是引用自一个对象	x is y，如果 id(x) 等于 id(y)，is 返回结果 1
is not	is not 是判断两个标识符是不是引用自不同对象	x is not y，如果 id(x) 不等于 id(y). is not 返回结果 1

2.3.5 位运算符

按位运算把数字转换为二进制数字来运算。Python 中的按位运算符有左移运算符(<<)、右移运算符(>>)、按位与(&)、按位或(|)和按位翻转(~)。位运算符如表 2.5 所示。

表 2.5　位运算符

运算符	名称	描　述	实　例
<<	左移	把一个数的比特向左移一定数目(每个数在内存中都表示为比特或二进制数字,即 0 和 1)	2<<2 得到 8——2 按比特表示为 10
>>	右移	把一个数的比特向右移一定数目	11>>1 得到 5——11 按比特表示为 1011,向右移动 1 比特后得到 101,即十进制的 5
&	按位与	数的按位与	5&3 得到 1
|	按位或	数的按位或	5|3 得到 7
^	按位异或	数的按位异或	5^3 得到 6
~	按位翻转	x 的按位翻转是−(x+1)	~5 得到 6

2.4　表达式

2.4.1　表达式组成

表达式是由数字、运算符和变量等构成的有意义的排列组合,通常由运算符号(操作符)和参与运算的数(操作数)两部分组成,经过运算后产生的运算结果类型由数据和运算符共同决定。

数学表达式转化为 Python 表达式应注意以下几点:

(1) 乘号不能省略。例如,x 乘以 y 写成 Python 表达式为 x * y。

(2) 括号必须成对出现,均使用圆括号,出现多个圆括号时,从内向外逐层配对。

(3) 运算符不能相邻。例如,a+−b 是错误的。

简单地说,将数学表达式转换为 Python 的表达式具有以下两种方法:

(1) 添加必要的运算符号,如乘号、除号等。

(2) 添加必要的函数,用于转换数学符号,例如,数学表达式 $\sqrt{25}$ 转换为 Python 表达式为 sqr(25)等。

【例 2-5】　数学表达式转换为 Python 表示式,如表 2.6 所示。

表 2.6　数学表达式转换为 Python 的表达式

数学表达式	Python 表达式
$\frac{abcd}{efg}$	a * b * c * d/e/f/g 或 a * b * c * d/(e * f * g)
$\sin45^\circ+\frac{e^{10}+\ln10}{\sqrt{x}}$	math.sin(45 * 3.14/180)+(math.exp(10)+math.log(10))/math.sqrt(x)
$[(3x+y)-z]^{1/2}/(xy)^4$	math.sqrt((3 * x+y)−z)/(x * y)^4

2.4.2 优先级

在一个表达式中,Python 会根据优先级由高到低进行运算。Python 运算符的优先级在表 2.7 中由上至下逐次降低。

表 2.7 运算符的优先级

运算符	描述	运算符	描述
lambda	Lambda 表达式	*,/,%	乘法、除法与取余
or	布尔"或"	+x,-x	正负号
and	布尔"与"	~x	按位翻转
not x	布尔"非"	**	指数
in,not in	成员测试	x.attribute	属性参考
is,is not	同一性测试	x[index]	下标
<,<=,>,>=,!=,==	比较	x[index:index]	寻址段
|	按位或	f(arguments...)	函数调用
^	按位异或	(experession,...)	绑定或元组显示
&	按位与	[expression,...]	列表显示
<<,>>	移位	{key:datum,...}	字典显示
+,-	加法与减法	'expression,...'	字符串转换

2.4.3 结合性

运算符通常由左向右结合,即具有相同优先级的运算符按照从左向右的顺序计算。例如,2+3+4 被计算成(2+3)+4。但是,也有个别运算符由右向左进行计算,例如,赋值运算符是由右向左结合,即 a=b=c 被处理为 a=(b=c)。

2.5 系统函数

内置函数(BIF,built-in functions)又称系统函数,或内建函数,是指 Python 本身所提供的函数,Python 常用的内置函数有数学函数、转换函数和随机数函数等。

2.5.1 数学函数

Python 中的数学函数包含在 Math 类中,如表 2.8 所示。

表 2.8　数学函数

函　数　名	描　述	实　例	结　果
abs(数值参数)	求绝对值	Abs(−38.5)	38.5
max(数值 1,数值 2)	求最大值	Max(3,2)	3
min(数值 1,数值 2)	求最小值	Min(3,2)	2
sum	求序列之和	sum([1,7,3])	11
sqrt(x)	开根号	math.sqrt(4)	2.0

2.5.2　转换函数

Python 提供了数据类型转换的函数，如表 2.9 所示。

表 2.9　常用的转换函数

函数名	描　　述	实　例	结　果
ord()	返回字符的 ASCII 编码	Ord('A')	65
chr()	返回指定编码的字符	Chr(97)	'a'
bin()	十进制数转换成二进制数	bin(4)	0b100
oct()	十进制数转换成八进制数	oct(8)	010
hex()	十进制数转换成十六进制数	hex(100)	0x64
int()	取整	int(−2.5) int(2.5)	−22
float(x)	将 x 转换到一个浮点数	float(2)	2.0
complex(real [,imag])	创建一个复数	complex(2,3)	(2+3j)
str()	数值转化成字符串	str(122.35)	"122.35"

【例 2-6】　转换函数举例。

```
>>> s= '15'
>>> s +1
Traceback (most recent call last):
    File "<interactive input>", line 1, in<module>
TypeError: cannot concatenate 'str' and 'int' objects
>>> int(s)+1
16
```

2.5.3　随机数函数

随机数可以用于数学、游戏和安全等领域中。Python 的随机数通过 random 模块中的 range()函数实现。

【例 2-7】　range()函数举例。

range()返回一个迭代器，根据需要生成一个指定范围的数字。有如下 3 种情况：

```
>>> range(1,5) # 代表从 1~5(不包含 5)
[1, 2, 3, 4]
>>> range(1,5,2) # 代表从 1~5,间隔 2(不包含 5)
[1, 3]
>>> range(5) # 代表从 0~5(不包含 5)
[0, 1, 2, 3, 4]
```

2.6 Python 字符

2.6.1 保留字符

保留字符又称关键字，Python 的关键字只包含小写字母，如表 2.10 所示。

表 2.10 Python 的保留字符

and	exec	not	def	if	return
assert	finally	or	del	import	try
break	for	pass	elif	in	while
class	from	print	else	is	with
continue	global	raise	except	lambda	yield

2.6.2 转义字符

python 转义字符同 C 语言的转义字符，采用反斜杠(\)，如表 2.11 所示。

表 2.11 Python 的转义字符

转义字符	描　述	转义字符	描　述	转义字符	描　述
\(在行尾时)	续行符	\e	转义	\f	换页
\\	反斜杠符号	\000	空	\oyy	八进制数，yy 代表的字符，例如：\o12 代表换行
\'	单引号	\n	换行	\xyy	十六进制数，yy 代表的字符，例如：\x0a 代表换行
\"	双引号	\v	纵向制表符	\other	其他的字符以普通格式输出
\a	响铃	\t	横向制表符		
\b	退格(Backspace)	\r	回车		

2.7 习题

一、选择题

1. 下面属于合法的变量名的是( )。

A. X_yz B. 123abc C. and D. X-Y

2. 下面属于不合法的整常数的是( )。

A. 100 B. &O100 C. &H100 D. %100

3. 表达式 16/4－2**5 * 8/4%5//2 的值为( )。

A. 14 B. 4 C. 20 D. 2

4. 数学关系表达式 3≤x<10 表示成正确的 Python 表达式为( )。

A. 3<=x<10 B. 3<=x and x<10

C. x>=3 or x<10 D. 3<=x and<10

5. 与数学表达式 ab/(3cd)对应,Python 的不正确表达式是( )。

A. a * b/(3 * c * d) B. a/3 * b/c/d

C. a * b/3/c/d D. a * b/3 * c * d

二、用 Python 表达式表示下列命题

(1) n 是 m 的倍数 (2) n 是小于正整数 k 的偶数

(3) x≥y 或 x<y (4) x,y 中有一个小于 z

(5) x,y 都小于 z (6) x,y 两者都大于 z,且为 z 的倍数

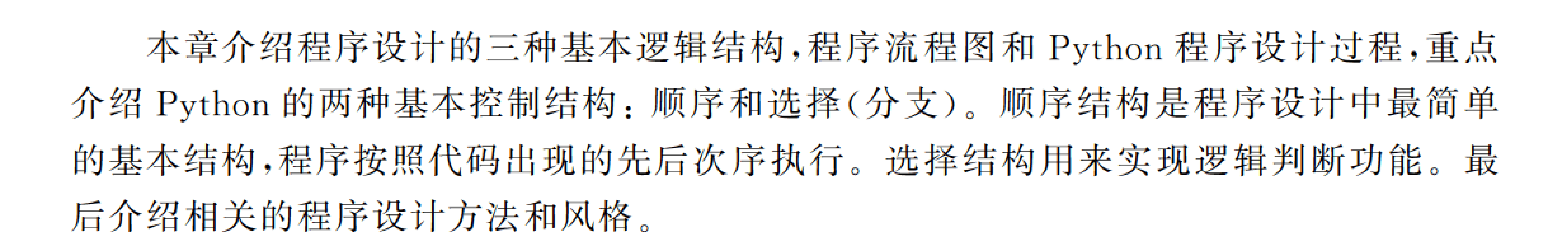
三、计算题

设 a=7,b=－2, c=4,求下列表达式的值

(1) a /2 * b/2－c (2) a % 3＋b * b－c//5

(3) 1234.5678 * a＋0.5 /100－b

第3章 顺序与选择结构



本章介绍程序设计的三种基本逻辑结构，程序流程图和 Python 程序设计过程，重点介绍 Python 的两种基本控制结构：顺序和选择（分支）。顺序结构是程序设计中最简单的基本结构，程序按照代码出现的先后次序执行。选择结构用来实现逻辑判断功能。最后介绍相关的程序设计方法和风格。

3.1 程序设计过程

3.1.1 三种基本逻辑结构

程序处理流程一般有输入、处理、输出（IPO，In-Process-Out）三大步骤，如图 3.1 所示。输入包括变量赋值、输入语句；处理包括算术运算、逻辑运算、算法处理等；输出包括打印输出、写入文件和数据库等。

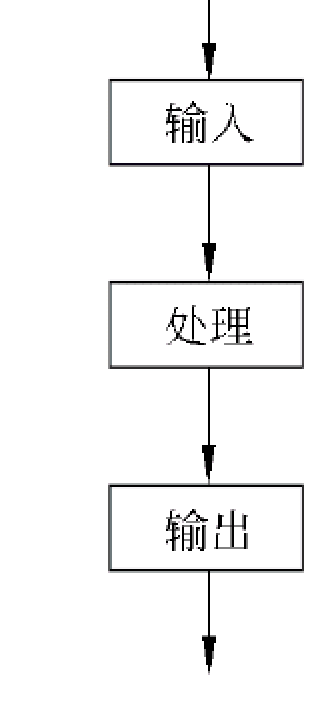


图 3.1 程序处理流程

1996 年，意大利人 Bobra 和 Jacopini 提出了三种基本结构：顺序结构、分支（选择）结构和循环结构。

顺序结构是最简单的控制结构，按照语句书写的顺序一句一句地执行。顺序结构的语句主要是赋值语句。例如火车在轨道上行驶，只有过了上一站点才能到达下一站点。

选择结构又称为分支语句、条件判定结构，它表示在某种特定的条件下选择程序中的特定语句执行，即对不同的问题采用不同的处理方法。例如在一个十字路口处，可以选择向东、南、西、北几个方向行走。

循环结构是指程序从某处开始有规律地反复执行某一操作块的现象。如果满足条件表达式后，反复执行某些语言或某一操作，就需要循环结构。循环结构作为程序设计中最能发挥计算机特长的基本结构，可以减少程序代码重复书写的工作量。例如 1000 米跑，围着足球场跑道不停地跑，直到满足条件（2.5 圈）时才停下来。

3.1.2 程序流程图

程序设计过程采用自然语言描述容易产生二义性，即歧义。例如，英文单词“doctor”，意为“博士”或“医生”，需要根据“doctor”所处的语境决定其含义，不同的语境有

不同的含义。在数学和计算机科学领域,往往采用流程图、伪语言、PAD 图和形式化语言(Z 语言等)描述算法,其中最普遍的是流程图。

程序流程图又称为框图,采用一些几何框图、流向线和文字说明表示算法,具有以下优点:

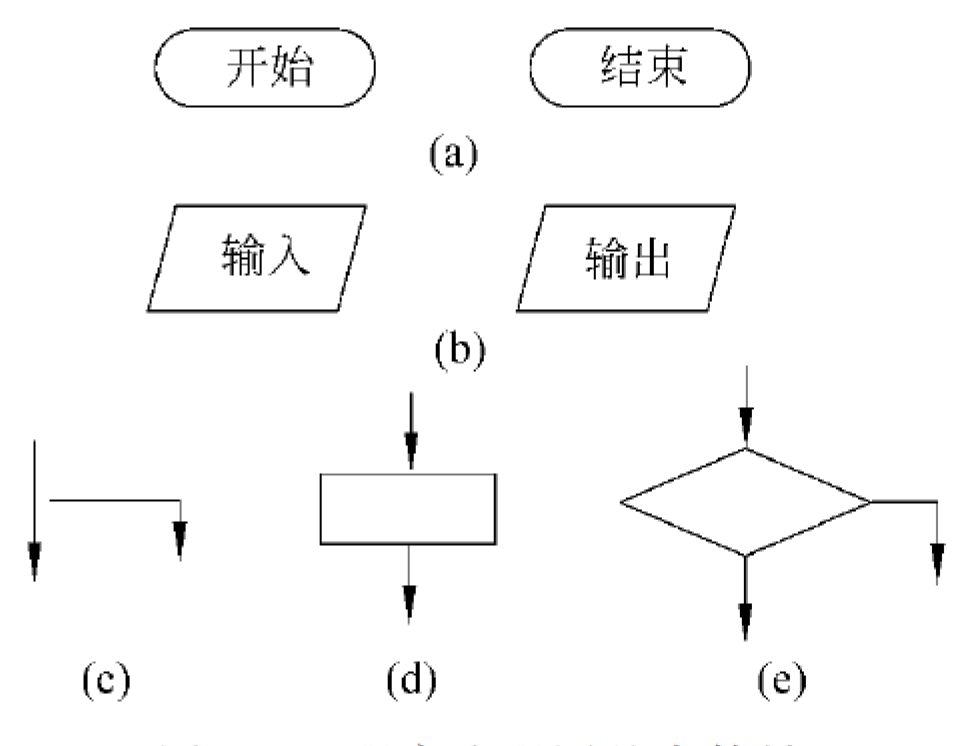


图 3.2　程序流程图基本符号

(1) 采用简单规范的符号,画法简单。

(2) 结构清晰,逻辑性强。

(3) 便于描述,容易理解。

程序流程图主要采用如下符号进行问题的描述:

(1) 开始框和结束框用于流程的开始和结束,如图 3.2(a)所示。

(2) 输入框向程序输入数据,输出框用于程序向外输出信息,如图 3.2(b)所示。

(3) 箭头表示控制流向,如图 3.2(c)所示。

(4) 执行框又称为方框,用于表示一个处理步骤,如图 3.2(d)所示,箭头是一进一出。

(5) 判别框又称为菱形框,用于表示一个逻辑条件,如图 3.2(e)所示,箭头是一进两出。

3.1.3　Python 程序设计流程

采用 Python 设计程序一般分为如下步骤,如图 3.3 所示。

步骤 1:分析找出解决问题的关键之处,即找出解决问题的算法,确定算法的步骤。

步骤 2:将算法转换为程序流程图,用于描绘对实际问题的解决步骤。

步骤 3:写程序:根据程序流程图编写 Python 代码。

步骤 4:调试程序:运行、纠正错误、修改程序、输入试验数据,观察结果。

步骤 5:运行程序:输入正确数据得到正确结果。

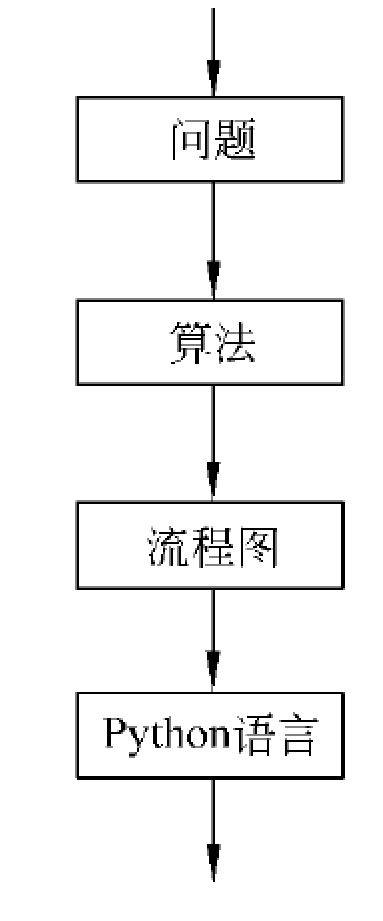


图 3.3　Python 程序设计流程

【例 3-1】 从键盘输入一个三位正整数,将其分解为个位、十位、百位三个整数。

步骤 1:分析。找出解决问题的关键之处。

步骤 2:画程序流程图。描绘出对实际问题的解决步骤,如图 3.4 所示。

步骤 3:写程序。根据程序流程图编写 Python 代码。

```
x=input('a number between 100~999: ')
a=x//100
b=(x-100*a)//10
c=x-100*a-10*b
print a, b, c
```

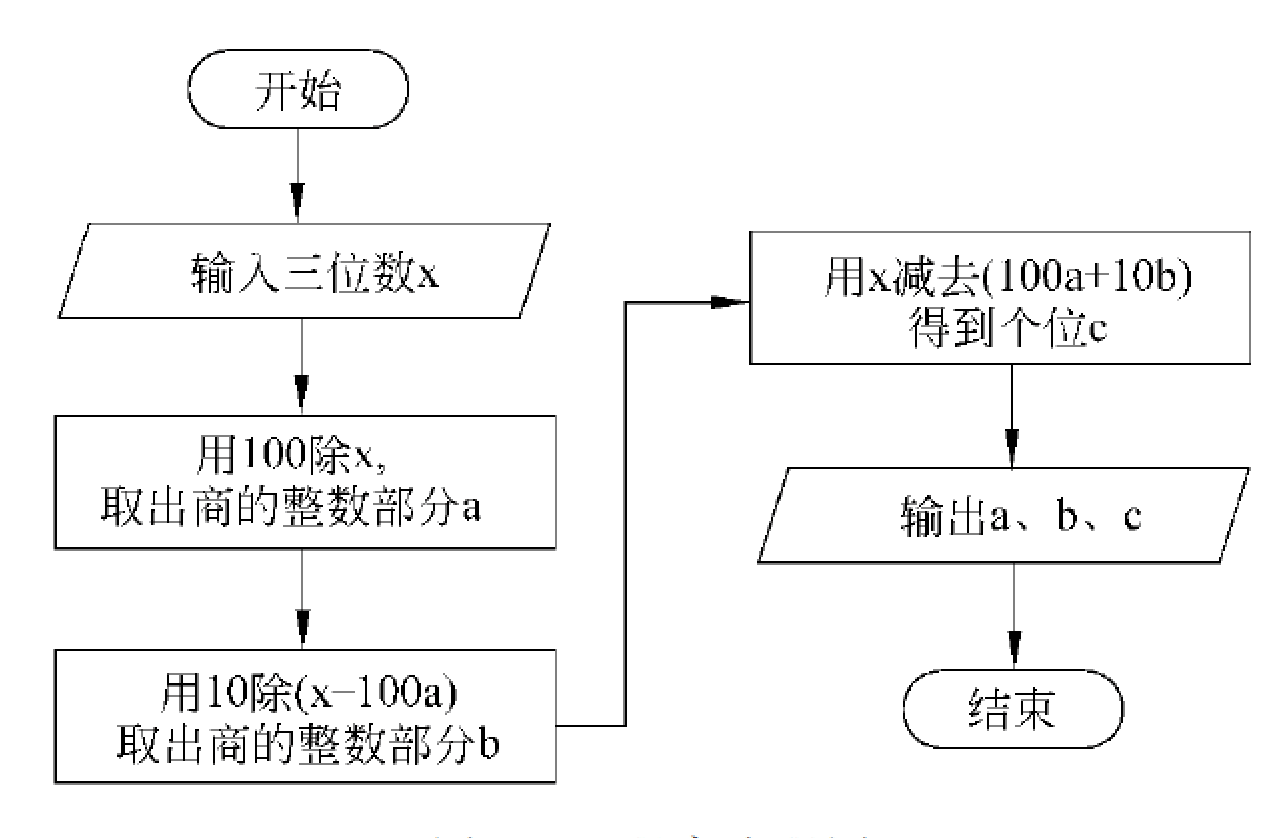


图 3.4 程序流程图

步骤 4：调试、测试程序。输入测试数据观察结果，例如，从键盘输入 123，输出结果 1 2 3。

步骤 5：运行程序。

3.2 代码书写规则

3.2.1 缩进

程序设计的风格主要强调“清晰第一，效率第二”，应注意程序代码书写的视觉组织。书写程序代码，如果所有语句都从最左一列开始，则很难清楚程序语句之间的关系，因此在程序书写过程中，如判断语句、循环语句等按一定的规则进行缩进处理，使得代码具有层次性并提高可读性。例如，C 语言中的缩进对于代码的编写来说是“有了更好”，而不是“没有不行”，仅作为书写代码风格。而 Python 语言将缩进作为语法严格要求。

C 语言与 Python 语言缩进对比如图 3.5 所示。

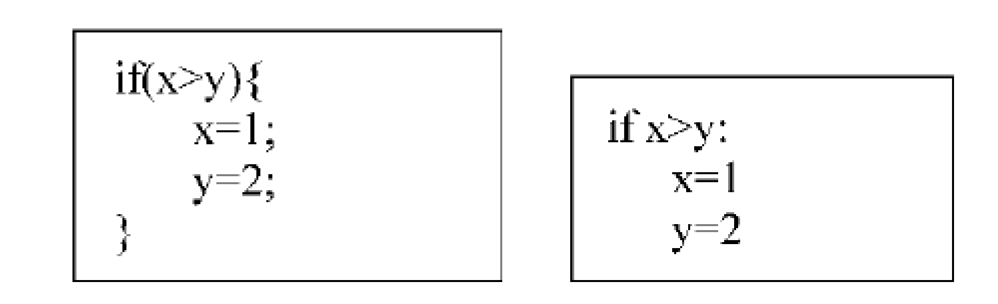


图 3.5 C 语言与 Python 语言缩进对比

Python 使用代码块的缩进来体现代码之间的逻辑关系，行首的空白称为缩进，缩进结束就表示一个代码块结束了。

Python 依靠这种缩进对代码进行解释和执行，具体如下：

(1) 在 Python 中的缩进等同于其他语言中的大括号。

(2) Python 利用行首的空白(空格和制表符 Tab 键)来决定逻辑行的缩进层次。

(3) 同一层次的语句必须有相同的缩进，每一组这样的语句称为一个块。

(4) Python 并未指定缩进的空白(空格和制表符)数目。

(5) 强烈建议在每个缩进层次使用单个制表符或两个或四个空格。

注意： 要么都是空格，要么都是 Tab 制表符，千万别混用。

3.2.2 逻辑行与物理行

物理行是 Python 在书写程序代码的表现形式。逻辑行是 Python 解释的代码形式。Python 希望物理行与逻辑行一一对应，每行都只使用一个语句，便于代码易读理解。但

是，Python 也满足代码书写灵活的要求，具体如下所示：

（1）Python 中每个语句以换行结束。

（2）一个物理行中使用多于一个逻辑行，即多条语句书写在一行，使用分号（;），例如：

```
principal=1000; rate=0.05; numyears=5;
```

（3）当语句太长时，也可以一条语句跨多行书写，即在多个物理行中写一个逻辑行，用反斜线（“\”）作为续行符。

【例 3-2】　反斜线（“\”）举例。

```
Print \
i
```

与如下写法效果相同：

```
print i
```

但是，当语句中包含[]，{ } 或 () 括号就不需要使用多行连接符。如下所示：

```
days=['Monday', 'Tuesday', 'Wednesday',
       'Thursday', 'Friday']
```

3.2.3　空行

函数之间或类的方法之间用空行分隔，表示一段新的代码的开始。类和函数入口之间也用一行空行分隔，以突出函数入口的开始。

空行与代码缩进不同，空行并不是 Python 语法的一部分。书写时不插入空行，Python 解释器运行也不会出错。但是空行的作用在于分隔两段不同功能或含义的代码，便于日后代码的维护或重构。

注意：空行也是程序代码的一部分。

3.2.4　注释

程序的注释分为序言性注释和功能性注释。注释一般位于所要注释代码的上一行，帮助读者思考每个过程、每个函数、每一条 Python 语句的意义，有利于程序的维护和调试。

（1）使用三重引号进行多行注释。

```
''' print(" I am in")
print ("I am in") '''
print("I am out")
```

运行结果：

```
I am out
```

（2）以“#”进行单行注释。

```
print "hello world"          #输出一行字
```

3.3　顺序结构程序设计

3.3.1　赋值语句

赋值号用“＝”表示，其左边只能是变量，不能是常量、常数符号、表达式，赋值语句不允许写成“表达式＝变量”，Python 中常用的赋值语句包括基本赋值语句、序列赋值、多目标赋值以及复合赋值等形式。

（1）基本赋值语句

【例 3-3】 基本赋值语句举例。

```
>>>x=1;y=2
>>>z=x+y
>>>print z
3
```

（2）序列赋值

【例 3-4】 序列赋值举例。

```
>>> (a,b)=(1,'zhou')
>>>print a,b
1 zhou
```

（3）多目标赋值

【例 3-5】 多目标赋值举例。

```
>>>i=j=k=3
>>>print i,j,k
3 3 3
```

（4）复合赋值

复合赋值运算符如表 3.1 所示。

表 3.1　复合赋值运算符

运算符	描　　述	实　　例
＋＝	加法赋值运算符	c＋＝a 等效于 c＝c＋a
－＝	减法赋值运算符	c－＝a 等效于 c＝c－a
*＝	乘法赋值运算符	c *＝a 等效于 c＝c * a
/＝	除法赋值运算符	c/＝a 等效于 c＝c/a
%＝	取模赋值运算符	c%＝a 等效于 c＝c%a
**＝	幂赋值运算符	c**＝a 等效于 c＝c**a
//＝	取整除赋值运算符	c//＝a 等效于 c＝c//a

【例 3-6】 复合赋值举例。

```
>>>x=1
>>>z=0
>>>z+=x
>>>print z
1
```

3.3.2 输入与输出

1. 数据输入

Python 提供 raw_input()和 input()两个函数实现数据输入。raw_input()接收字符串类型的输入数据。input()要求输入数据必须是数值类型。

【例 3-7】 数据输入举例。

```
>>>raw_input_A=raw_input("raw_input: ")
raw_input: abc
>>>raw_input_A
'abc'
>>>input_A=input("Input: ")
  Input: abc
    Traceback (most recent call last):
      File "<pyshell# 11>", line 1, in<module>
        input_A=input("Input: ")
    File "<string>", line 1, in<module>
    NameError: name 'abc' is not defined
```

2. 数据输出

Python 通过 print 函数实现数据的输出操作，print 函数的语法结构如下所示：

```
print  <expression>,<expression>
```

print 的操作对象是字符串。

注意：

(1) 在 Python 命令行下，print 是可以省略的，默认就会输出每一次命令的结果。

(2) 多个＜expression＞间用逗号间隔。

(3) 格式化控制输出，其作用和 C 语言中的 printf()函数基本相同，具体如下所示：

① %d 用来替换整数。

② %f 用来替换小数。

- %f 格式符选项对应一个十进制浮点数，不指定精度时打印 6 位小数。
- 使用包含“.2”精度修正符的%%f 格式符选项将只打印 2 位小数。

③ %s 用来替换一段字符串。

【例 3-8】 数据输出举例。

```
>>>print 'Price is % f' %  4.99
Price is 4.990000
>>>print 'Price is % .2f' %  4.99
Price is 4.99
#保留两位小数,在 f 前面加上条件: % .2f
>>>print 'Today is % s.' %  'Friday'
Today is Friday.
```

print 会自动在行末加上回车,自动换行。如果不需换行输出,只须在 print 语句的结尾添加逗号“,”。

【例 3-9】 换行举例。

```
for i in range(0,3):
    print i
```

输出结果如图 3.6 所示。

```
for i in range(0,3):
    print i,
```

输出结果如图 3.7 所示。



图 3.6 例 3-9 程序运行结果 1



图 3.7 例 3-9 程序运行结果 2

3.3.3 顺序结构

顺序结构是最简单的控制结构,按照语句书写先后次序依次执行。顺序结构的语句主要是赋值语句、输入与输出语句,其特点是程序沿着一个方向进行,具有唯一的入口和出口。如图 3.8 所示,顺序结构只有先执行完语句 1,才会执行语句 2,语句 1 将输入数值进行处理后,其输出结果作为语句 2 的输入。也就是说,如果没有执行语句 1,语句 2 不会执行。

【例 3-10】 从键盘上输入一整数作为半径,求圆的面积和周长。

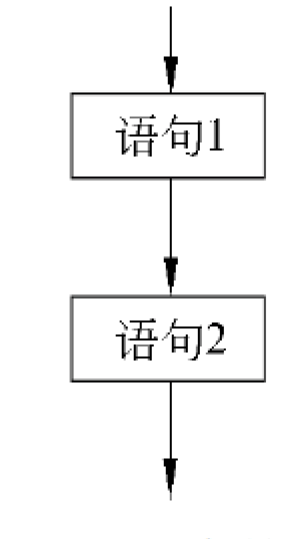


图 3.8 顺序结构图

```
PI=3.1415926
number=input("please in put a number")
```

```
area= PI * number * number
Perimeter=2 * PI * number * number
print"circle's Perimeter is ",Perimeter
print "circle's area is ",area
```

程序运行结果如图 3.9 所示。

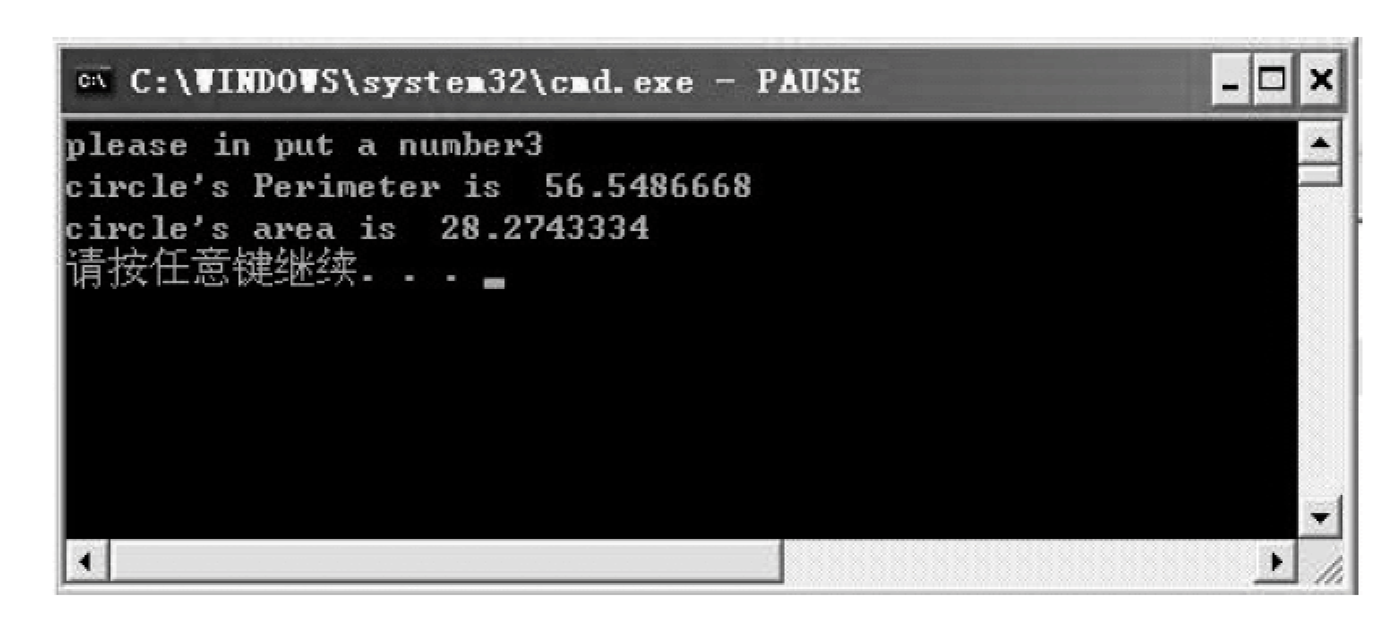


图 3.9　例 3-10 程序运行结果

3.4　选择结构程序设计

选择结构又称为分支语句、条件判定结构，表示在某种特定的条件下选择程序中的特定语句执行。条件一般有以下几种表达式组成：

(1) 关系表达式：是指由一个或多个变量结合关系运算符组成的表达式。

(2) 逻辑表达式：逻辑表达式又称为布尔表达式，它是指由一个或多个关系表达式结合布尔运算符组成的表达式。

(3) 算术表达式：是指由算术运算符构成的表达式，其结果为数值。在 Python 中，将“零”数值看成 false，将非零数值看成 true。

Python 是通过 if 语句来实现分支结构。if 语句具有单分支、双分支和多分支等形式。

3.4.1　单分支

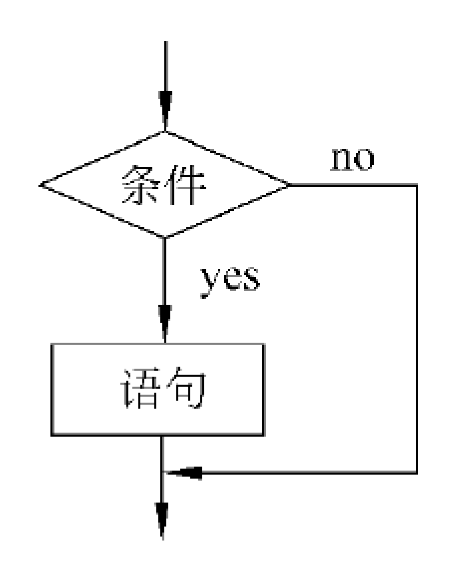
if 的单分支语句流程图如图 3.10 所示。

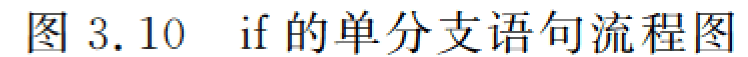
```
if  条件表达式:
    语句块
```

【例 3-11】 从键盘上输入两个正整数 x 和 y，升序输出。

【解析】 若从键盘依次输入的两个数是 3 和 5，只须顺序输出两个数。但若输入的先后次序是 5 和 3，则必须对两个数交换后输出。不妨设两个整数为 x 和 y，引入临时变量 t，通过以下三步实现 x 和 y 的交换，如图 3.11 所示。

x 和 y 交换过程如表 3.2 所示。




图 3.10 if 的单分支语句流程图

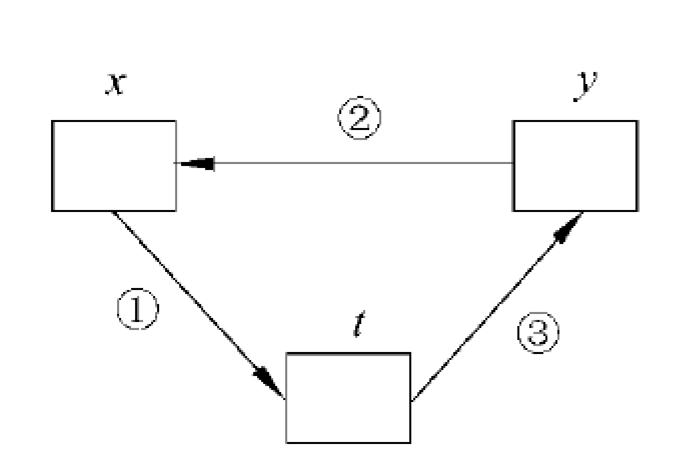
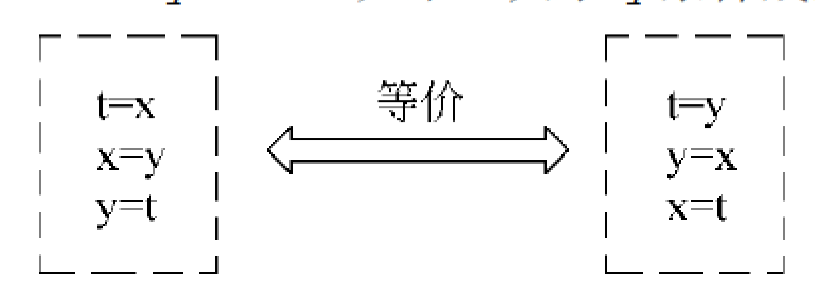


图 3.11 x 和 y 交换,引入临时变量 t

表 3.2 交换变量图示

交换步骤	变量 *x*	变量 *y*	变量 *t*
交换前	5	3	0
步骤一	5	3	5
步骤二	3	3	5
步骤三	3	5	5

```
x=input("please input x")
y=input("please input y")
print "exchange before:", x, y
if x>y:      #如果 x 大于 y 条件成立,则引入 t 交换 x 和 y
```




```
print "exchange after", x, y
```

程序运行结果如图 3.12 所示。



图 3.12 程序运行结果

【例 3-12】 从键盘上输入三个整数,按照从小到大的升序排序。

【解析】 三个变量 x,y,z 通过排列组合取值共有 6 种输入顺序。不妨设 x,y,z 输入为降序排列,即 x 为这三个数的最大数,则有 $x>y$ 且 $x>z$,需分别进行 x 与 y 和 x 与 z 的交换,使得 x 成为 x,y,z 中最小数,然后将 y 和 z 交换,使得 $y<z$。故整个排序需要进行三次交换,执行三次 if 语句。

3.4.2 双分支

if 语句的双分支流程图如图 3.13 所示。当条件表达式的值为 true 时，程序执行语句 1；当条件表达式的值为 false 时，程序执行语句 2。

if 的双分支语句书写格式如下：

```
if  条件表达式:
    <语句块 1>
else:
    <语句块 2>
```

图 3.13 if 语句的双分支流程图

if 和 else 的语句块用缩进来表示，else 从句在某些情况下可以省略。

【例 3-13】 输出 a 与 b 中较大的数。

```
a=input("a=")
b=input("b=")
if a<b:
    z=b
else:
    z=a
print z
```

3.4.3 多分支

当分支超过 2 个时，采用 if 语句的多分支语句。该语句的作用是根据不同的条件表达式的值确定执行哪个语句块，Python 测试条件依次为条件表达式 1、条件表达式 2…，当某个条件表达式值为 true 时，就执行该条件下的语句块，其余分支不再执行；若所有条件都不满足，且有 else 子句，则执行 else 语句块，否则什么也不执行。

if 的多分支语句格式如下所示：

```
if 条件表达式 1 :
    <语句块 1>
elif  条件表达式 2 :
    <语句块 2>
elif  条件表达式 3:
    <语句块 3>
 ⋮
else:
    <语句块 n>
```

if 语句的多分支流程图如图 3.14 所示。

【例 3-14】 实现向用户显示问候信息的功能，根据时间的不同，问候信息也不同。根据当前时间是上午、下午或晚上，分别给出不同的问候信息。

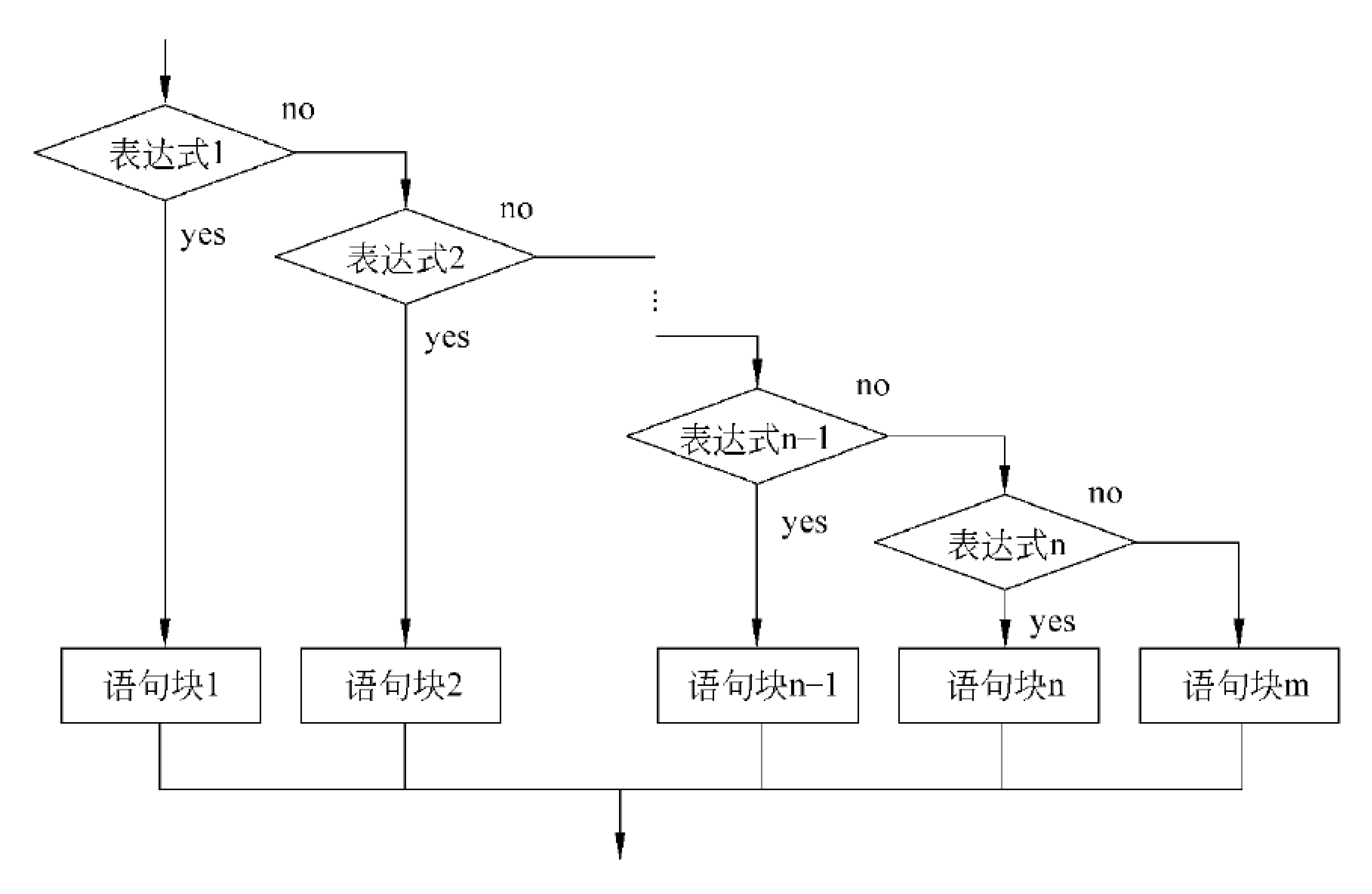


图 3.14　if 语句的多分支流程图

```
hour=input("hour")
if hour<=12:
    print"Good morning"
if (hour>12) and (hour<18):
    print"Good afternoon"
if hour>=18:
    print "Good Evening"
```

采用并列三条 if 的单分支语句实现此功能，这三条 if 语句按照顺序结构依次执行。如果当前时间小于 12，则第 1 条 if 语句的判断条件 hour<=12 为真，执行"Good morning"，之后对第 2 条和第 3 条 if 语句进行判断执行。而在这种情况下，第 2 条和第 3 条 if 语句已经没有必要继续执行，因此，三条 if 语句的并列使用虽然可以实现功能，但效率较差。改为 if 语句的多分支结构则可以避免以上情况，如下所示。

```
hour=input("hour")
if hour<=12 :
    print "Good morning"
elif hour<18:
    print "Good afternoon"
else:
    print "Good Evening"
```

3.4.4　选择结构嵌套

在一个分支的语句块中，继续进行新的分支。这种情况称为选择(分支)结构的嵌套。嵌套的形式如下：

```
if 表达式 1:
    语句块 1
⋮
    if 表达式 11:
        语句块 11…
        else:
        语句块 12
    ⋮
else:
    语句块 2
```

【例 3-15】 输入三角形的三边长，判断是否能组成三角形；若可以构成三角形，则输出它的面积和三角形类型(等腰、等边、直角、普通)。

【解析】 三角形构成条件是任意两边之和大于第三边。三角形类型判断：①等腰的判断条件为两边相等；②等边的判断条件为三边均相等；③直角三角形可通过勾股定理来判断；④不满足以上条件的就是普通三角形。程序流程图如图 3.15 所示。

```
import math
a=input("please input a")
b=input("please input b")
c=input("please input c")
if a>b:
    t=a;a=b;b=t
if a>c:
    t=a;a=c;c=t
if b>c:
    t=b;b=c;c=t
if a+b>c and a+c>b and b+c>a:                    #两边之和大于第三边
    s=(a+b+c)/2.0
    area=math.sqrt(s*(s-a)*(s-b)*(s-c))          #用海伦公式计算三角形面积
    print """triangle's area is """,area
    if a==b and a==c:                            #等边三角形
        print "triangle is Equilateral "
    elif a==b or a==c or b==c:                   #等腰三角形
        print "triangle is isosceles"
    elif a*a+b*b==c*c:                           #直角三角形
        print "triangle is right-angled "
    else:
        print "triangle is normal"               #普通三角形
else:
    print "No triangle"
```

程序运行结果如图 3.16～图 3.20 所示。

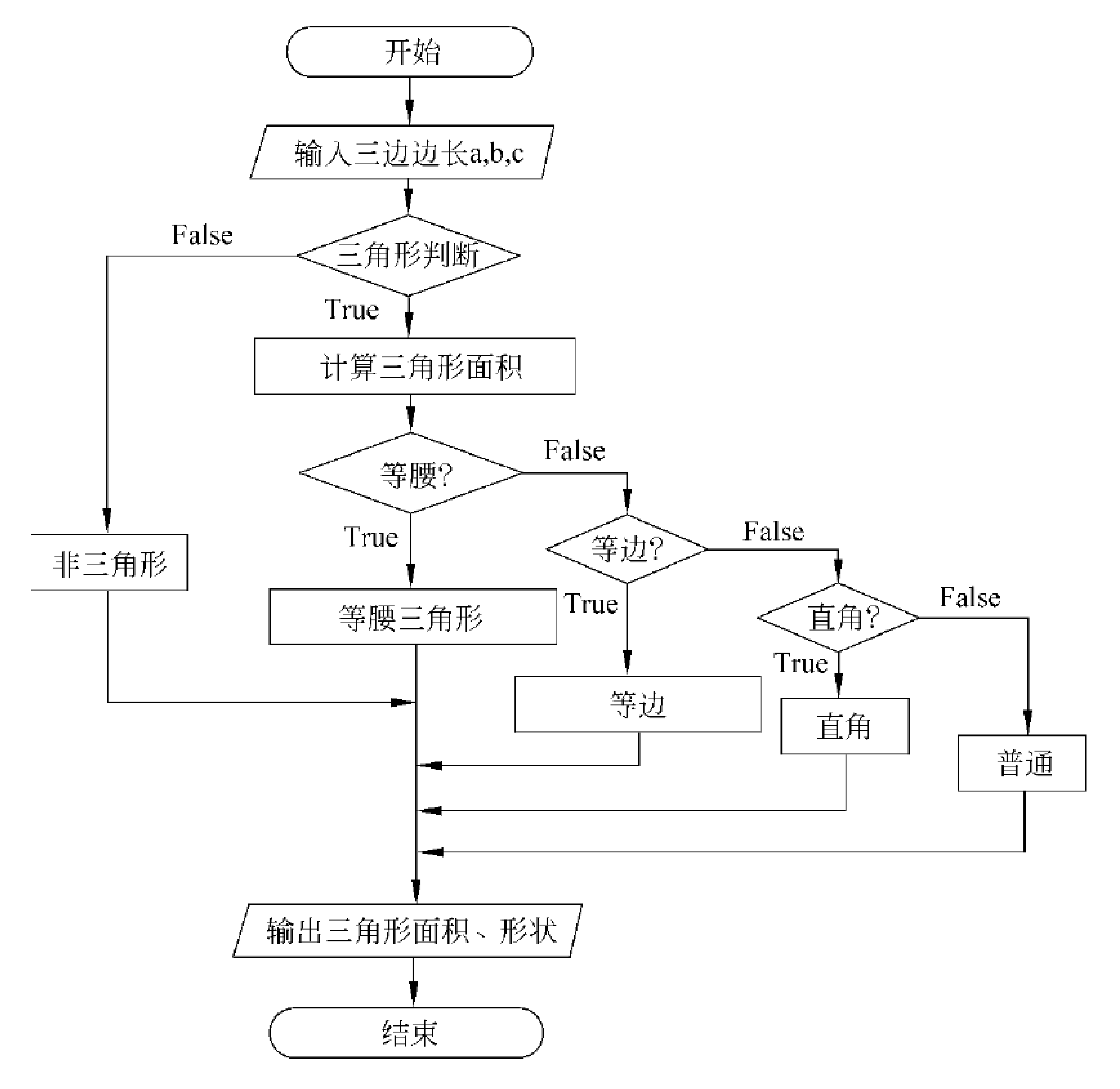


图 3.15　例 3-15 程序流程图

```
C:\WINDOWS\system32\cmd.exe - PAUSE
please input a2
please input b4
please input c3
triangle's area is  2.90473750966
triangle is normal
请按任意键继续. . .
```

图 3.16　一般三角形

```
C:\WINDOWS\system32\cmd.exe - PAUSE
please input a2
please input b1
please input c3
No triangle
请按任意键继续. . .
```

图 3.17　不是三角形

```
C:\WINDOWS\system32\cmd.exe - PAUSE
please input a3
please input b5
please input c4
triangle's area is  6.0
triangle is  right-angled
请按任意键继续. . .
```

图 3.18　直角三角形

```
C:\WINDOWS\system32\cmd.exe - PAUSE
please input a6
please input b4
please input c6
triangle's area is  11.313708499
triangle is  isosceles
请按任意键继续. . .
```

图 3.19　等腰三角形

```
C:\WINDOWS\system32\cmd.exe - PAUSE
please input a6
please input b6
please input c6
triangle's area is  15.5884572681
triangle is Equilateral
请按任意键继续. . .
```

图 3.20　等边三角形

3.5　程序设计方法与风格

3.5.1　语句构造方法

下面列出一些良好的语句构造方法，方便程序的编辑和调试。

(1) 在一行内只写一条语句。

(2) 稍复杂的表达式中要积极使用括号，以免造成优先级的混乱以及二义性。

(3) 单个函数的程序行数最好不要超过 100 行。

(4) 尽量使用系统函数。

(5) 不要随意定义全局变量，尽量使用局部变量。

(6) 保持注释与代码完全一致。

(7) 循环、分支层次最好不要超过 5 层。

(8) 尽量减少使用“否定”条件语句。

(9) 对输入数据进行合法性检验。

3.5.2 编程规范

Python 语言的学习过程就是发现错误，改正错误的过程，首先应不断积累每一个 Python 的知识点，将其反复实践掌握；其次，编程涉及很多知识，如操作系统、软件工程、数据结构、面向对象程序设计、硬件系统等，需要不断扩充知识面。

下面介绍一些编程规范。

(1) 养成良好的编程习惯

Python 程序设计的入门学习并不难，但养成良好的编程习惯却是一个十分重要的过程。

① 上机实践前先在纸上构思程序设计思路。

② 每次上机后应及时总结，把没有搞清楚的问题记录下来，进行分析与总结。

③ 平时应多抽课余时间，多上机调试程序。

④ 注意系统的提示信息，特别是错误信息的提示。

(2) 多写程序、注重实践

程序设计课是高强度的脑力劳动，必须动手编写程序才能学会。从小的程序设计开发开始，逐渐提高开发程序的规模。只有在编写大量程序之后，通过一定量的积累，才能发生质变，动手能力的培养是编程语言学习的核心。

(3) 阅读、借鉴别人设计好的程序

多阅读别人设计好的程序代码，可以按以下步骤进行：

① 要读懂别人的程序。

② 思考能不能将程序加以修改，完成更多的功能。

(4) “实例学习”

通过学习、编写生动有趣的小例子，掌握 Python 编程的知识点和小技巧。

(5) 经常使用帮助文件

Python 帮助文档如图 3.21 所示。

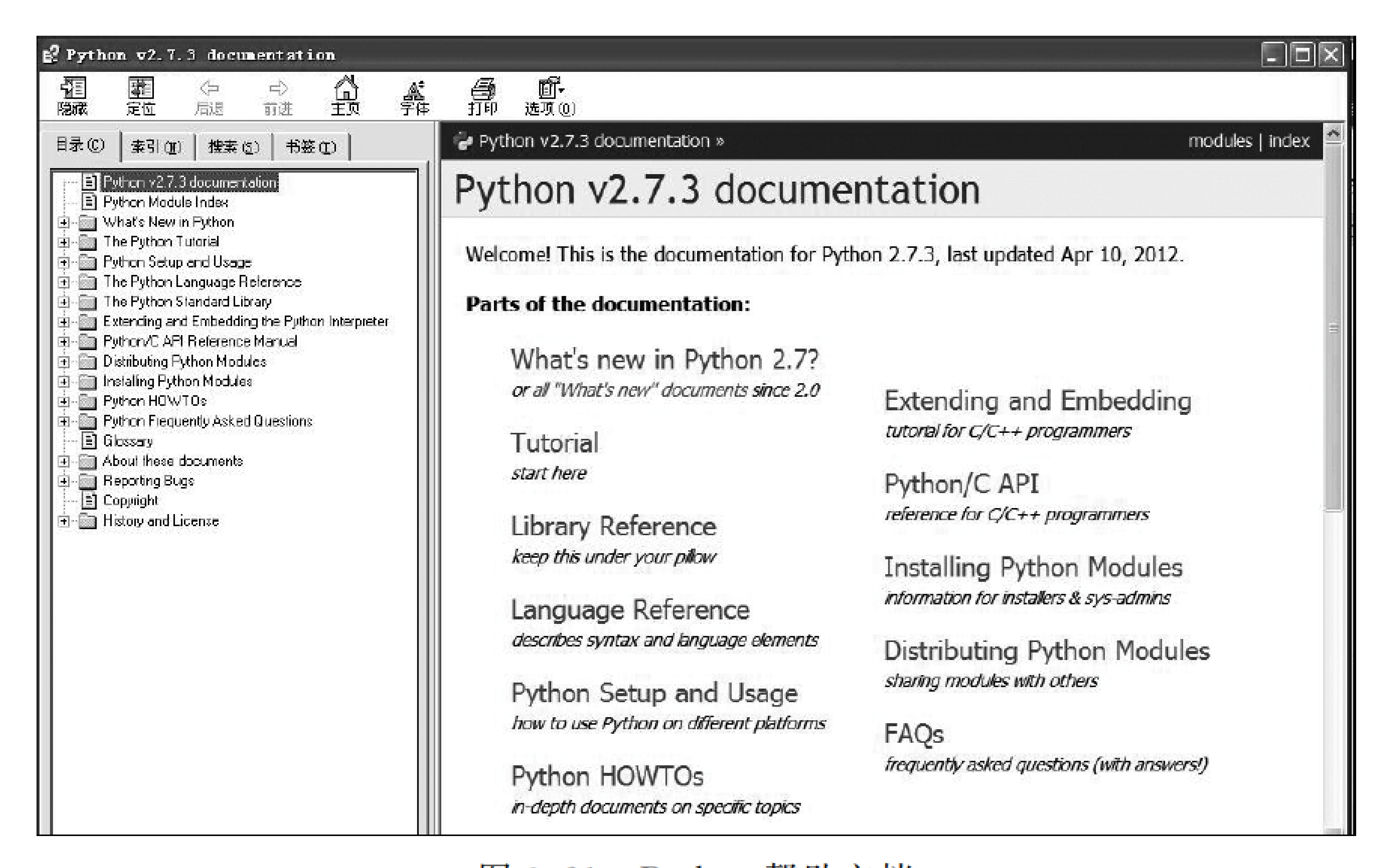


图 3.21 Python 帮助文档

3.6 习题

一、选择题

1. 在一条语句内写多条语句时，每个语句之间用(　　)符号分隔。

   A. ，　　B. ：　　C. 、　　D. &

2. 一条语句要在下一行继续写，用(　　)符号作为续行符。

   A. ＋　　B. －　　C. _　　D. ；

二、编程题

1. 编写一个程序：从键盘输入某个时钟的分钟数，将其转化为用小时和分钟表示。

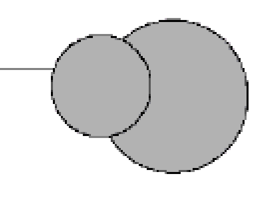
2. 在购买某物品时，若标明的价钱 x 在下面范围内，所付钱 y 按对应折扣支付，其数学表达式如下：

$$y=\begin{cases}x, x<100\\0.9x, 1000\leqslant x<2000\\0.8x, 2000\leqslant x<3000\\0.7x, x>3000\end{cases}$$

3. 编写一个程序：判断用户输入的字符是数字字符、字母字符还是其他字符。

4. 编写计算圆面积和球体积的程序。要求输出结果只保留四位小数；如果半径的输入不合法，如含有非数值字符，提示错误，并在错误信息得到用户确认之后，重新输入计算。

第4章



循环结构

循环结构是指程序有规律地反复执行某一操作块的现象。本章介绍 Python 语言的两种循环语句：while 循环和 for 循环。while 循环常用于多次重复运算，而 for 循环用于遍历序列型数据。最后介绍相关的辅助语句以及循环嵌套。

4.1 循环

4.1.1 循环引入

【例 4-1】 求 1～5 五个数之和。

```
i=1
sum=0
sum+=i;i+=1
sum+=i;i+=1
sum+=i;i+=1
sum+=i;i+=1
sum+=i
print sum
```

【解析】 sum+=i;i+=1 重复写了 5 次，题意若改为 1～100 之和，则 sum+=i;i+=1需要写上 100 遍，工作量极大。为此，对于这种重复有规律的行为，Python 引入了循环结构。

4.1.2 循环概述

在许多实际问题中，需要对问题的一部分通过若干次有规律的重复计算来实现。例如，求大量数据之和、迭代求根，递推法求解等，这些都要用到循环结构。

在 Python 语言中，用于循环结构的流程控制语句有 while 和 for 两种循环结构。

4.2 while 语句

循环语句由循环体及循环控制条件两部分组成。反复执行的语句或程序段称为循环体。循环体能否继续执行，取决于循环控制条件。

while 语句的书写格式如下：

【格式一】

```
while 循环控制条件:
    循环体
```

【格式二】

```
while 循环控制条件:
    循环体
else:
    语句
```

while 语句流程图如图 4.1 所示。

while 语句的执行过程如下所示：

将初值赋给循环变量，判断循环变量的当前值是否超过终值，如果没有超过终值，执行循环体，循环变量增加一个步长值，返回进行条件判断；如果循环变量的当前值仍没有超过终值，则继续执行循环体，循环变量再增加一个步长值，返回再进行条件判断。如此反复，直到循环变量的当前值超过终值，也就是条件表达式的结果为假，结束循环，不再执行循环体。

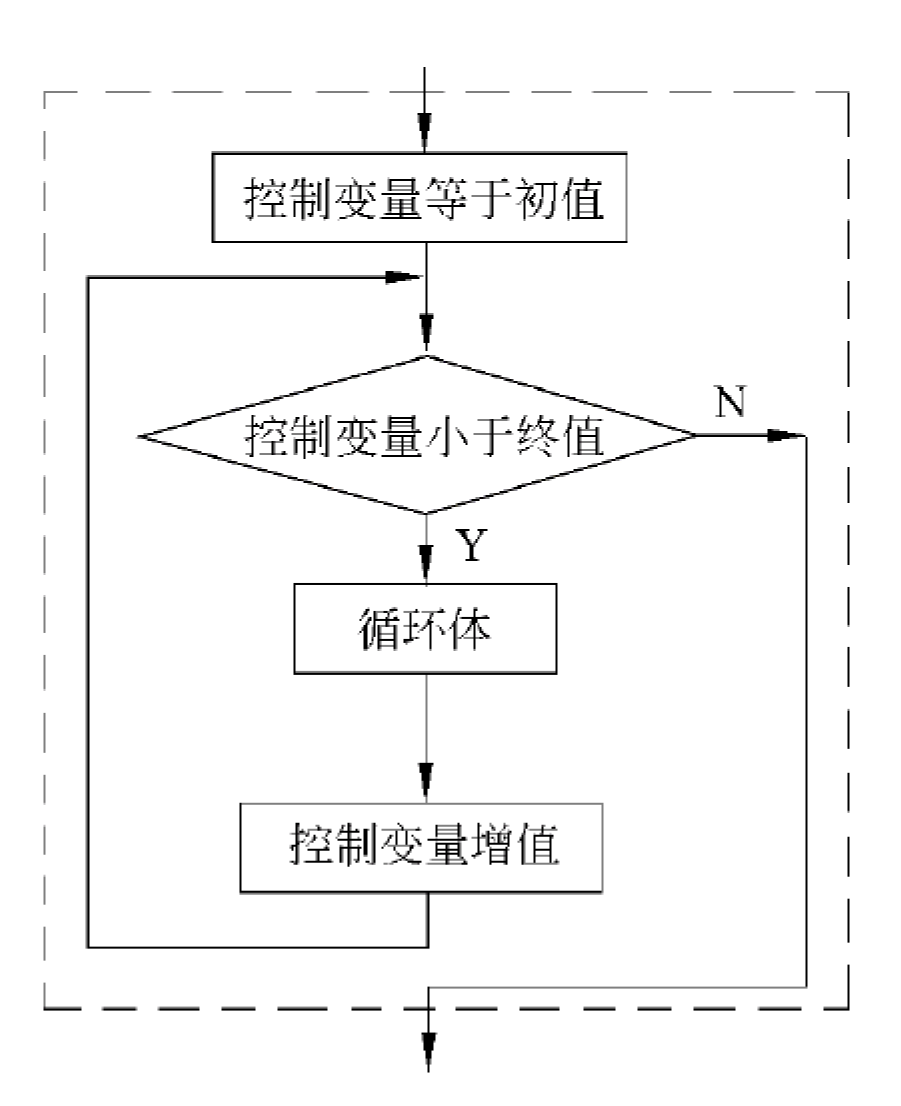


图 4.1　while 语句循环流程图

Python 按以下步骤执行 while 循环：

步骤 1：将＜循环变量＞设置为＜初值＞。

步骤 2：若＜步长＞为正数，则判断＜ 循环变量＞是否小于＜终值＞，若不是，退出循环，否则继续下一步。若＜步长＞为负数，则判断＜循环变量＞是否小于＜终值＞，若是，退出循环，否则继续下一步。

步骤 3：执行循环体语句。

步骤 4：＜循环变量＞的值增加＜步长＞值。

步骤 5：返回步骤 2。

4.2.1　确定次数循环

循环分为确定次数循环和不确定次数循环。确定次数循环是指在循环开始之前就可以确定循环体执行的次数。

【例 4-2】　计算 1～100 之间所有自然数的和。

【解析】　题意为 1＋2＋3＋4＋…＋100，这种求取一批数据的“和”的计算称为“累加”，是一种典型的循环。通常引入一个变量存放“部分和”(Sum)；另一个变量表示变化的量，即累加项(i)。设置 sum 的初值为 0，然后通过循环重复执行：和值＝和值＋累加项。

```
i=1;sum=0                #i为循环变量,sum表示累加的和
while i<=100:            #从 1 到 100
    sum=sum+i            #部分和累加
    i+=1                 #每次步长为 1
print "sum",sum          #总和
```

循环结构的构造关键是确定与循环控制变量(i)有关的表达式 1(i＝1)、表达式 2(i<＝100)和表达式 3(i＋＝1),具体解释如下:

表达式 1(i＝1):循环控制变量初值,作为循环开始的初始条件。

表达式 2(i<＝100):用于判断是否执行循环体的条件。当满足表达式 2 时,循环体被执行,反之,当条件表达式 2 的结果为假,则退出循环体,不再反复执行。设想,如果条件表达式 2 始终为真,循环体将会反复一直地被执行,不会停止,就会产生"死循环"。那么如何让循环终止呢?也就是说,如何让表达式 2 条件判断的结果为假,从而终止循环呢?于是产生了表达式 3。

表达式 3(i＋＝1):循环控制变量变化。每次循环中,循环体执行一次,表达式 3 也执行一次,循环控制变量 i 增加步长值 1。经过 100 次循环,i 的值最终变为 101,表达式 2(i<＝100)的条件判断的结果为假,从而循环终止。也就是说,表达式 3 的作用是使得表达式 2 不成立,终止循环。

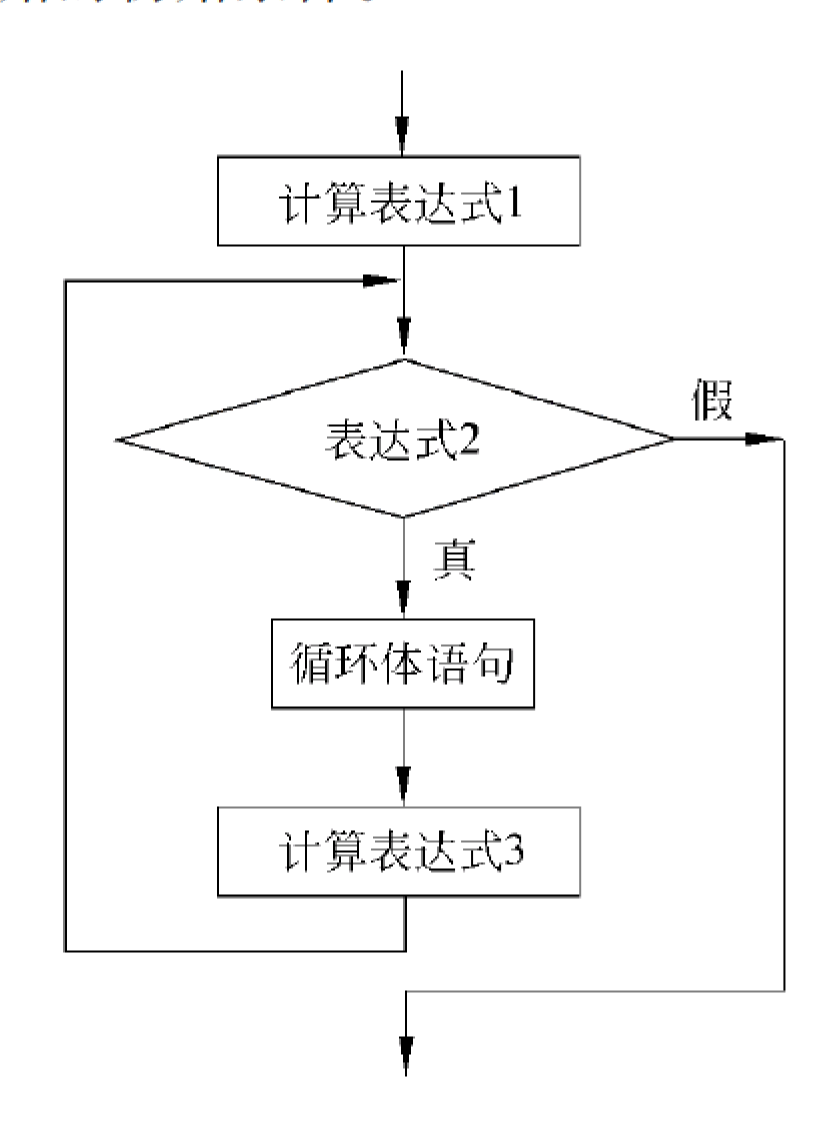


图 4.2　while 循环结构

while 循环中三个表达式的示意图如图 4.2 所示。

循环控制变量 i 的值如表 4.1 所示。

表 4.1　循环控制变量 i 值图示

循环控制变量 i	表达式 2 (i<＝100)	是否执行循环体	循环体 Sum＝sum＋i	表达式 3 (i＝i＋1)
0	true	执行	0	1
1	true	执行	1	2
2	true	执行	3	3
3	true	执行	6	⋮
⋮	⋮	执行	⋮	⋮
99	true	执行	4950	100
100	true	执行	5050	101
101	false	不执行	5050	101

【例 4-3】　计算 1～100 之间所有奇数的和。

【解析】　方法一:改变循环控制变量的表达式 3,即改变步长量。

```
i=1;sum=0
while i<=100:
    sum=sum+i
    i+=2                    #步长为 2
print "sum",sum
```

【解析】 方法二：对遍历的 i 进行判断，如果是奇数，进行累加。

```
i=1;sum=0
while i<=100:
    if i%2!=0:
        sum=sum+i
    i=i+1
print "sum",sum
```

【例 4-4】 输入 5 个整数，求出其平均值并输出结果。

【解析】 使用计数器(counter)来控制循环的执行次数，当 counter 超过 5，便停止循环。变量 total 用于存储部分和，初始化为 0。

```
total=0
counter=1
while counter<=5:
    total=total+int(raw_input('input a number'))
    counter=counter+1
print "avager is " ,float(total)/5
```

运行结果如图 4.3 所示。



图 4.3 例 4-4 程序运行结果

4.2.2 不确定次数循环

不确定次数循环是指有些循环只知道循环结束的条件，而循环体所重复执行的次数事先并不知道。

【例 4-5】 求 π 的近似值，要求其误差小于 0.0000001。

【解析】 计算圆周率 π 的公式：$\pi/4=1-1/3+1/5-1/7+\cdots+1/n$，这个公式实际是一个数列的前 n 项之和，其和是在每次累加之后得到，无法事先确定循环的次数。

```
dblDenominator=1.0                  #用于存放项的分母
dblTerm=1.0                         #用于存放当前项的值
intSign=1                           #用于表示当前项的符号
Pai=0                               #用于存放累加和结果
while abs(dblTerm)>0.00001:
    Pai+=dblTerm
    dblDenominator+=2               #分母是第-1项的分母加上 2
    intSign=-intSign                #改变符号
    dblTerm=intSign/dblDenominator  #生成下一项
Pai=Pai * 4
print Pai
```

【例 4-6】 求两个整数的最大公约数、最小公倍数。

【解析】 辗转相除法又称为欧几里德算法，基本思路如下所示。

步骤 1：对于已知两数 m 和 n，使得 $m>n$。

步骤 2：m 除以 n 得余数 r。

步骤 3：若 $r=0$，则 n 为最大公约数结束；否则执行步骤 4。

步骤 4：n 赋值给 m，r 赋值给 n，再重复执行步骤 2。

```
m=input("a number")
n=input("a number")
nm=n * m
if m<n:
    t=m;m=n;n=t
    r=m%n
while r!=0:
    m=n
    n=r
    r=m%n
print "The Greatest common divisor=", n
print "The least common multiple=" ,nm/n
```

程序运行结果如图 4.4 所示。

```
C:\WINDOWS\system32\cmd.exe - PAUSE
a number6
a number8
The Greatest common divisor  = 2
The least common multiple = 24
请按任意键继续. . .
```

图 4.4　例 4-6“辗转相除法”运行结果

【例 4-7】 求以下表达式的值，其中 n 值从键盘输入(参考值：当 $n=11$，$s=1.833333$)。

$$s = 1 + \frac{1}{1+2} + \frac{1}{1+2+3} + \cdots + \frac{1}{1+2+3+\cdots+n}$$

【解析】 数学中的级数求和问题是循环结构解决的一类常见问题，其重点在于通过观察表达式的规律，分析出“通式”。该题的通式有以下几个：

(1) 分母的通式：mu=mu+i

(2) 变量 i 的通式：$i=i+1$

(3) 当前项的通式：t=1.0/mu

(4) 求和的通式：$s=s+t$

```
i=1
mu=0
s=0.0
n=input('请输入 n 值：')
while i<=n:                    #判断是否计算到表达式的最后一项
    mu=mu+i                    #求分母的通式
    i+=1                       #i 自增的通式
    t=1.0/mu                   #求当前项的通式
    s=s+t                      #求和的通式
print 's=', s                  #循环结束后，打印总和
```

【例 4-8】 求学生分数的平均值。

方法一：for 循环

```
n=input("How many numbers do you have? ")
sum=0.0
for i in range(n):
    x=input("Enter a number>>")
    sum=sum+x
print "\nThe average is", sum/n
```

输出

```
How many numbers do you have? 5
Enter a number>>32
Enter a number>>45
Enter a number>>34
Enter a number>>76
Enter a number>>45
The average of the numbers is 46.4
```

【解析】 方法一需要用户输入确定的循环次数 n，不适合事先不知道 n 的场合，引入不确定的条件循环 while 进行改进。

方法二：while 循环

```
flag='y'
sum=0.0
```

```
counter=0
while flag=='y':
    x=input("Enter a number>>")
    sum=sum+x
    counter=counter+1
    flag=raw_input("enter? (y or n)? ")
print "\nThe average is", sum/counter
```

程序运行结果如图 4.5 所示。

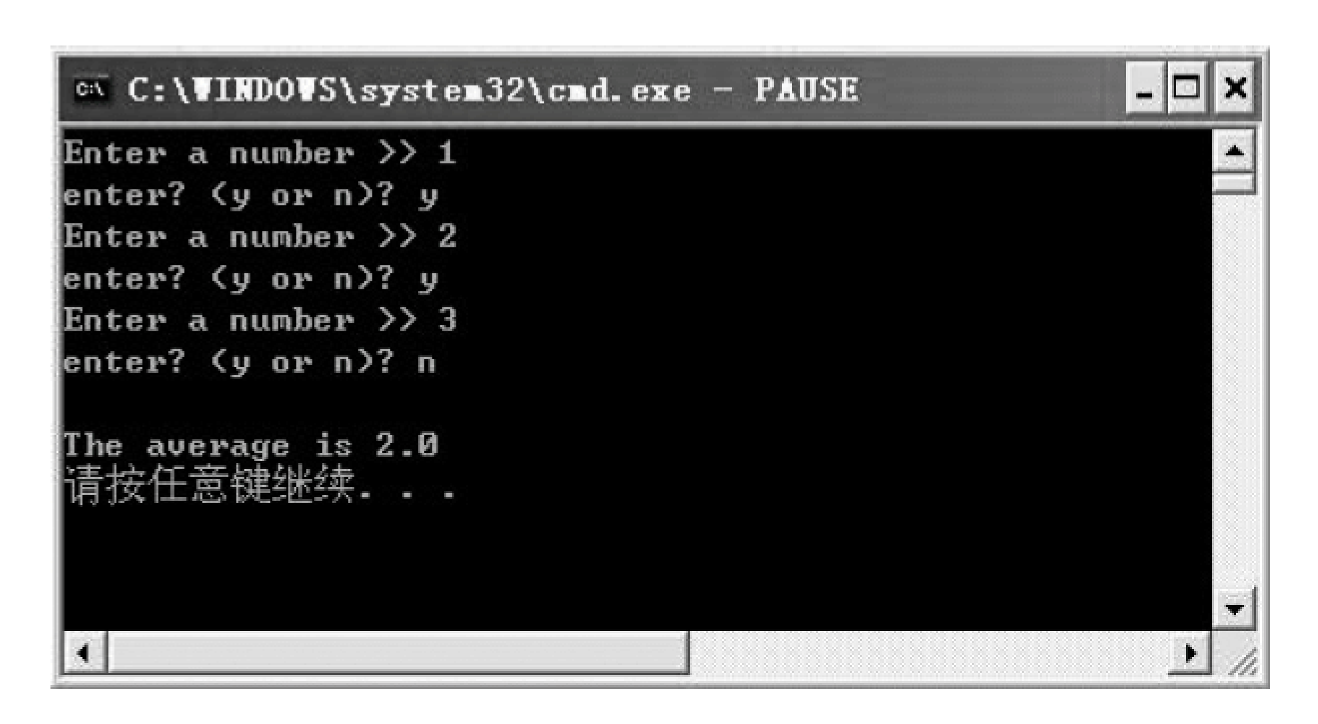


图 4.5　程序运行结果

【解析】 方法二设置一个是否继续循环的标志(flag),用户不用事先输入循环次数 n,由程序自己计算输入的值的个数。但是用户需要一直输入"y",改进方式是引入设置一个特殊数据值作为终止循环的信号。

方法三:信号值循环控制法

【解析】 信号值又称哨兵,是一个特殊值,用于指示循环结束。本题求学生分数的平均值,哨兵可设为小于 0 的数,循环将一直循环到哨兵才结束。

```
sum=0.0
count=0
x=input("Enter a number (negative to quit)>>")
while x>=0:
    sum=sum+x
    count=count+1
    x=input("Enter a number (negative to quit)>>")
print "\nThe average of the numbers is", sum/count
```

输出

```
Enter a number (negative to quit)>>32
Enter a number (negative to quit)>>45
Enter a number (negative to quit)>>34
Enter a number (negative to quit)>>76
Enter a number (negative to quit)>>45
Enter a number (negative to quit)>>-1
The average of the numbers is 46.4
```

4.2.3　无限循环

无限循环又称为死循环，当 while 语句的“表达式”永远为真，循环将永远不会结束。使用 while 语句构成无限循环的格式通常为

```
while True:
    循环体
```

一般采用在循环体内使用 break 语句强制结束死循环。

【例 4-9】　猜测数值。如果猜测结果不对，一直猜测，直到猜对为止。

```
number=1
running=True                  #表达式 1
while running:                #表达式 2
    guess=int(raw_input('Enter an integer : '))
    if guess==number:
        print 'Congratulations, you guessed it.'
        running=False         #猜对了，使得表达式 2 为布尔假，终止循环
    elif guess<number:
        print 'No, it is a little lower than the number'
    else:
        print 'No, it is a little higher than the number'
else:
    print 'The while loop is over.'
print 'Done'
```

输出结果如图 4.6 所示。



图 4.6　例 4-9 程序运行结果

4.3　for 语句

for 语句用于遍历型循环，依次对某个序列中全体元素进行处理。Python 的 for 语句与其他程序设计语言(如 Visual Basic 程序设计语言、C 程序设计语言)的 for 语句不太一样，Python 的 for 语句主要用于列表、元组等序列结构。

for 语句书写格式如下：

```
For 目标标识符 in 序列:
    循环体
```

【例 4-10】 for 循环应用于列表序列。

```
fruits=['banana', 'apple', 'mango']        #列表
for fruit in fruits:                       #Second Example
    print 'fruits have :', fruit
```

输出如图 4.7 所示。

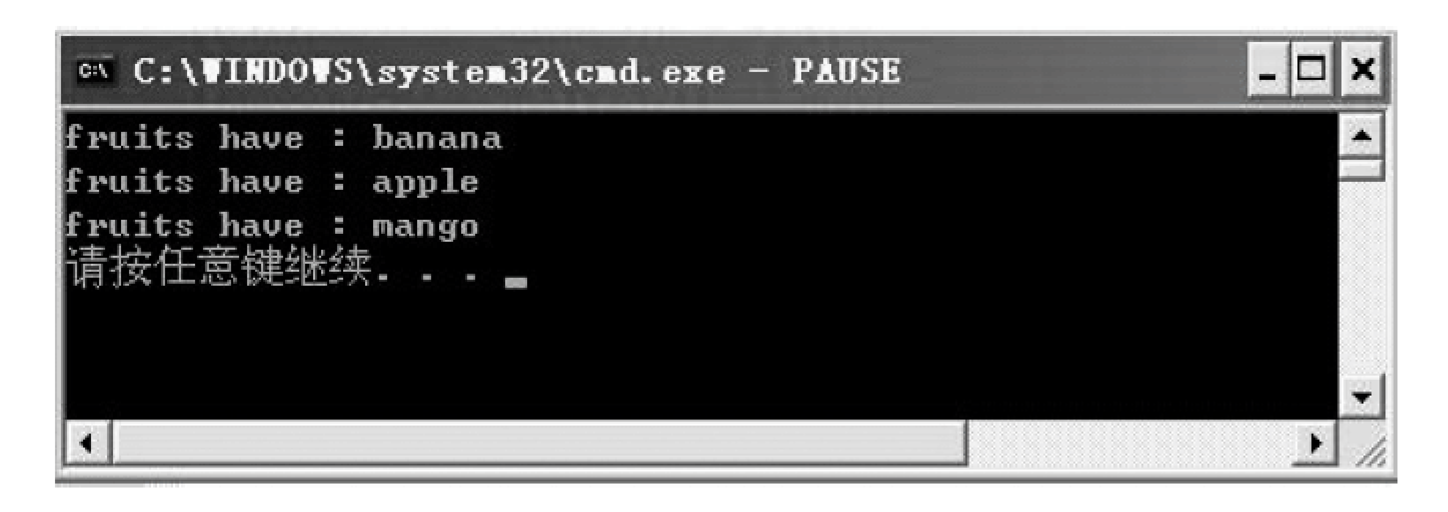


图 4.7 例 4-10 程序运行结果

【例 4-11】 求出分数列表的平均分。

【解析】 方法一：使用 Python 的内建函数 sum()求和，然后再求平均分。

```
>>>score=[70, 90, 78, 85, 97, 94, 65, 80]
>>>score
[70, 90, 78, 85, 97, 94, 65, 80]
>>>aver=sum(score)/8.0
>>>aver
82.375
```

【解析】 方法二：使用 for 语句进行遍历，程序流程图如图 4.8 所示。

(1) 列表 score 有 8 个元素，序列索引范围是 0～7。

(2) 成员测试运算符 in，如果成员在序列中，测试结果为 true，否则为 false。

(3) len()用于计算序列长度。

(4) range()给出循环取值的范围。

```
score=[70, 90, 78, 85, 97, 94, 65, 80]
print '所有的分数值是：'
print score                    #打印列表
sum=0
#以下 for 语句的 i 是迭代项，内建函数 len
(score)的执行结果是 8
```

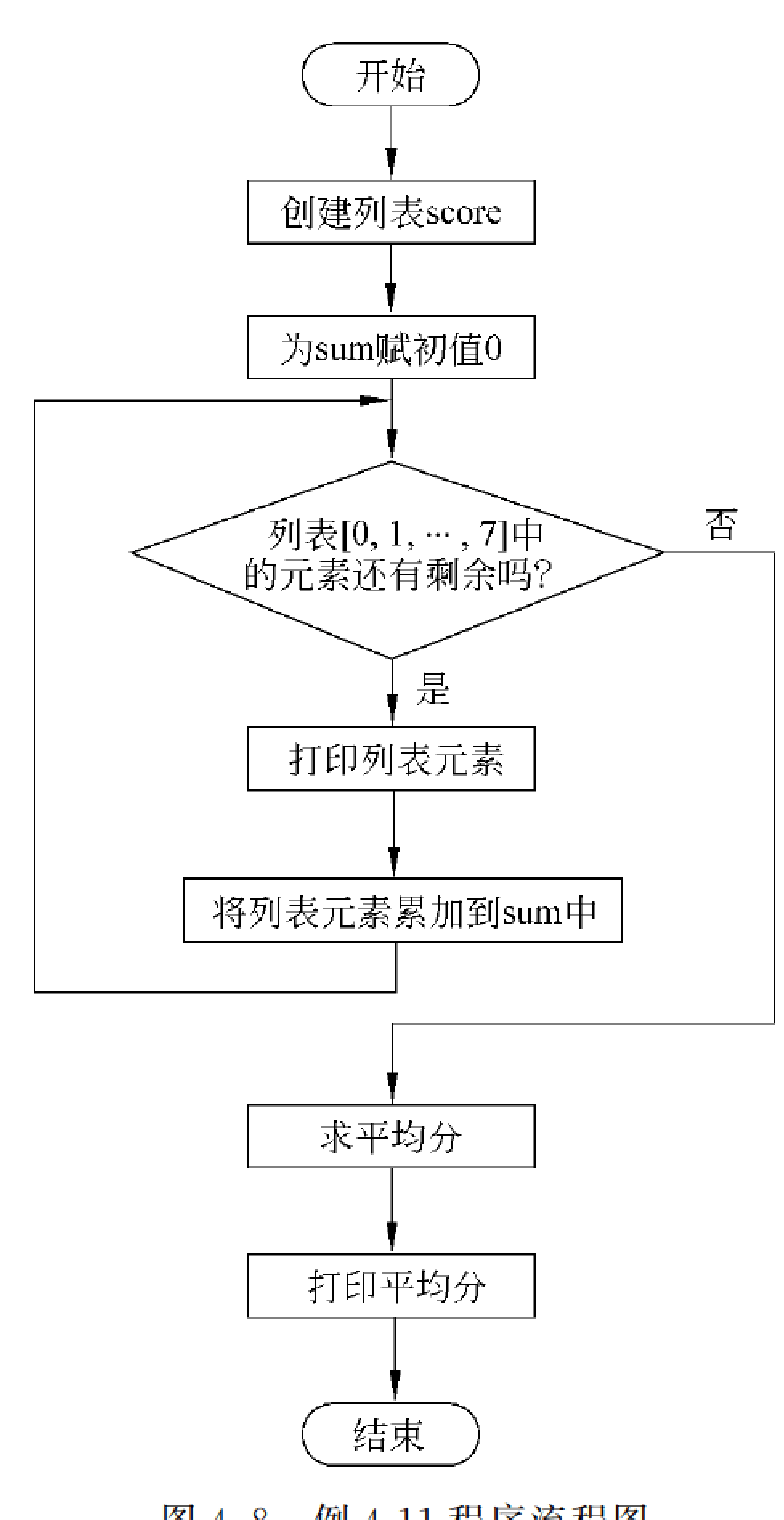


图 4.8 例 4-11 程序流程图

```
#内建函数 range(8)返回一个列表[0,1,2,3,4,5,6,7]
#运算符 in 是判断迭代项 i 是否还在列表[0,1,2,3,4,5,6,7]中
for i in range(len(score)):
    sum+=score[i]          #循环体——对列表元素求和
aver=sum/8.0               #循环之后,求平均值
print '\naver=', aver
```

【解析】 方法三：使用 for 语句，通过序列项运算。

```
score=[70,90,78,85,97,94,65,80]
print '所有的分数值是：'
print score                #输出列表
sum=0
#以下 for 语句使用运算符 in 判断迭代项 i 是否在列表 score 中
for i in score:
    sum+=i                 #循环体——对列表元素求和

aver=sum/8.0               #循环之后,求平均值
print '\naver=', aver
```

【例 4-12】 for 循环输出 1～100 的所有整数。

```
for i in range(1, 101):
    print i,
```

输出如图 4.9 所示。

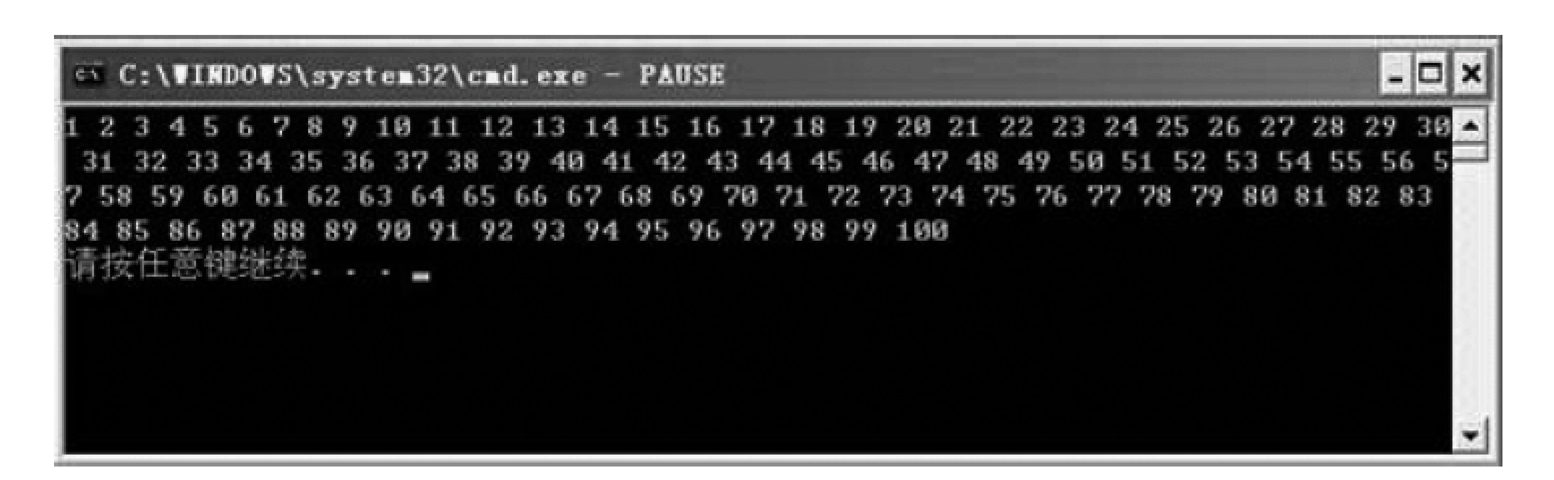


图 4.9　例 4-12 程序运行结果

4.4　辅助语句

之前的例子，都是与循环变量值超出终值范围或者循环条件不满足时，才终止循环。除此之外，还可以在循环体中使用 break 语句和 continue 语句，用于跳出循环控制结构。

4.4.1　break 语句

break 语句用于提前终止整个循环体的执行，与 C 语言中的 break 功能相同，打破了最小封闭 for 或 while 循环。在 while 和 for 的循环嵌套中，break 语句将停止执行最深层的循环，并开始执行下一行代码。

break 语句对循环控制的影响如图 4.10 所示。

说明：

① break 语句只能出现在循环语句的循环体中。

② 在循环语句嵌套使用的情况下，break 语句只能跳出(或终止)它所在的循环，而不能同时跳出(或终止)多层循环。

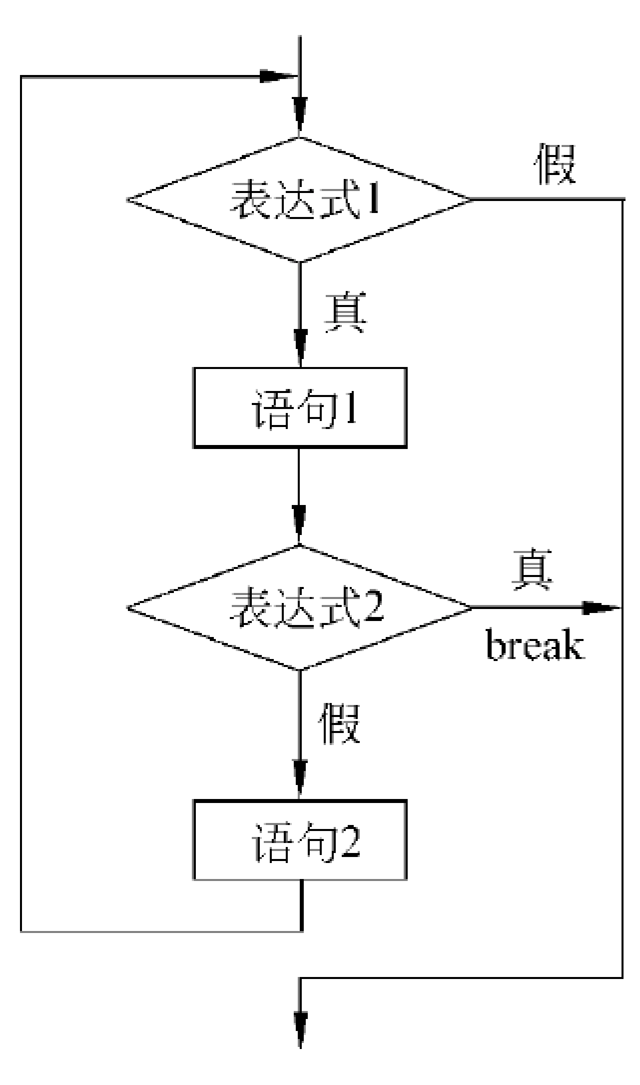


图 4.10　break 语句对循环控制的影响

【例 4-13】 求 2+4+6+8+…+n<100 成立的最大的 n 值。

【解析】 遍历过程以递增的方式进行，当找到第一个能使此不等式成立的 n 值，循环过程立即停止，后续还没有遍历的数无须再进行判断，可使用 break 语句将循环提前终止。

```
i=2; sum=0
while True:
    sum+=i
    if sum>=100:
        break
    else:
        i+=2
print i
```

4.4.2　continue 语句

continue 用于终止本次循环的执行，即跳过当前这次循环中的 continue 语句后尚未执行的语句，接着进行下一次循环条件的判断。

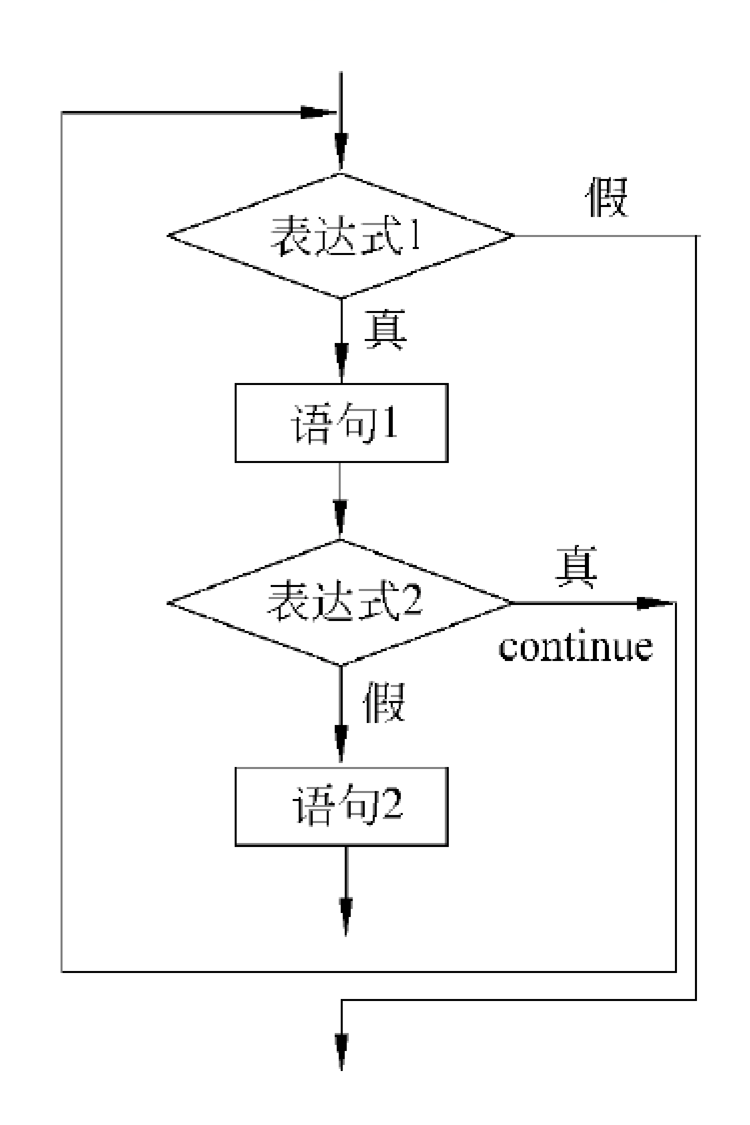


图 4.11　continue 语句对循环控制的影响

说明：

① continue 语句只能出现在循环语句的循环体中。

② continue 语句往往与 if 语句联用。

③ 若执行 while 语句中的 continue 语句，则跳过循环体中 continue 语句后面的语句，直接转去判别下次循环控制条件；若 continue 语句出现在 for 语句中，则执行 continue 语句就是跳过循环体中 continue 语句后面的语句，转而执行 for 语句的表达式 3。

continue 语句对循环控制的影响如图 4.11 所示。

【例 4-14】 求 200 以内能被 17 整除的所有正整数。

```
print '''Less than 200 numbers is divisible
by 17:'''
for i in range(1, 201, 1):
    if i%17!=0:
        continue
    print i,
```

程序运行结果如图 4.12 所示。

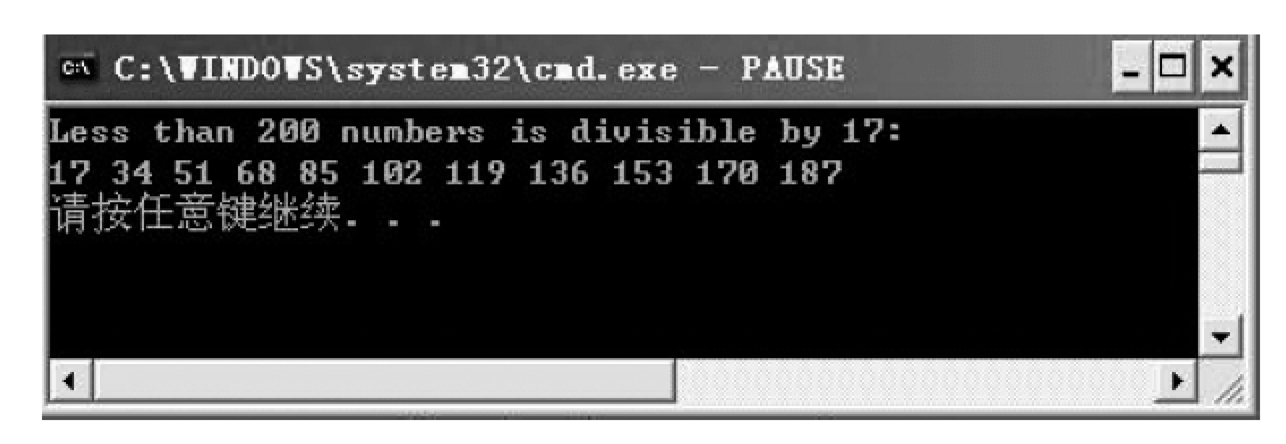


图 4.12　例 4-14 程序运行结果

4.4.3　else 语句

当 while 结构中存在可选部分 else，其循环体执行结束后，会执行 else 语句块（不管 while 里面）是否执行。但是，当 break 语句和 else 语句结合，while 循环因为 break 语句而终止，else 部分也不会被执行。

【例 4-15】

```
b=input('input a number')
a=b//2
while a>1:
    if b%a==0:
        print b,' is not prime'     #b 不是素数
        break
    a=a-1
else:
    print b,' is prime'
```

4.4.4　pass 语句

当某个子句不需任何操作，可使用 pass 语句保持程序结构的完整性。

【例 4-16】

```
if a<b:
    pass               #Do nothing
else:
    z=a
```

【例 4-17】

```
for letter in 'Python':
    if letter=='h':
        pass                    #若将 pass 换为 continue，程序运行结果是什么？
        print 'This is pass block'
    print 'Current Letter :', letter
print "Good bye!"
```

输出结果如图 4.13 所示。



图 4.13 例 4-17 程序运行结果

4.5 循环嵌套

在一个循环体里面嵌入另一个循环，这种情况称为多重循环，又称为循环的嵌套，较常使用的是二重循环。Python 语言允许在 while 循环中嵌入 for 循环，反之亦可。

Python 循环嵌套语法如下所示：

```
while expression:
    for iterating_var in sequence:
        statements(s)
    statements(s)
```

二重循环构造包括外层循环和内层循环，以及内外层之间的关系。一般情况，从单重循环出发，确定其中一个循环变量为定值，实现单重循环；然后改变此循环变量，将其从定值改为变量，将单重循环转变为双重循环。

【例 4-18】 打印九九乘法表。

【解析】 九九乘法表涉及乘数 i 和被乘数 j 两个变量，变化范围为 1～9。先假设被乘数 j 的值不变，假设为 1，实现单重循环。

```
for i in range(1,10):
    j=1
    print i ,"* " ,j , "=" ,i * j
```

程序运行结果如图 4.14 所示。

【解析】 将被乘数 j 的定值 1 改为变量，让其从 1～9 之间取值。

```
for i in range(1,10):
    for j in range(1,10):
        print i ,"* " ,j , "=" ,i * j,
    print
```

程序运行结果如图 4.15 所示。

【解析】 九九乘法表，乘数和被乘数满足交换律，故可列出一半的乘法表。

```
C:\WINDOWS\system32\cmd.exe - PAUSE
1 * 1 = 1
2 * 1 = 2
3 * 1 = 3
4 * 1 = 4
5 * 1 = 5
6 * 1 = 6
7 * 1 = 7
8 * 1 = 8
9 * 1 = 9
请按任意键继续. . .
```

图 4.14　例 4-18 程序运行结果之一

```
C:\WINDOWS\system32\cmd.exe - PAUSE
1 * 1 = 1 1 * 2 = 2 1 * 3 = 3 1 * 4 = 4 1 * 5 = 5 1 * 6 = 6 1 * 7 = 7 1 * 8 = 8 1 * 9 = 9
2 * 1 = 2 2 * 2 = 4 2 * 3 = 6 2 * 4 = 8 2 * 5 = 10 2 * 6 = 12 2 * 7 = 14 2 * 8 = 16 2 * 9 = 18
3 * 1 = 3 3 * 2 = 6 3 * 3 = 9 3 * 4 = 12 3 * 5 = 15 3 * 6 = 18 3 * 7 = 21 3 * 8 = 24 3 * 9 = 27
4 * 1 = 4 4 * 2 = 8 4 * 3 = 12 4 * 4 = 16 4 * 5 = 20 4 * 6 = 24 4 * 7 = 28 4 * 8 = 32 4 * 9 = 36
5 * 1 = 5 5 * 2 = 10 5 * 3 = 15 5 * 4 = 20 5 * 5 = 25 5 * 6 = 30 5 * 7 = 35 5 * 8 = 40 5 * 9 = 45
6 * 1 = 6 6 * 2 = 12 6 * 3 = 18 6 * 4 = 24 6 * 5 = 30 6 * 6 = 36 6 * 7 = 42 6 * 8 = 48 6 * 9 = 54
7 * 1 = 7 7 * 2 = 14 7 * 3 = 21 7 * 4 = 28 7 * 5 = 35 7 * 6 = 42 7 * 7 = 49 7 * 8 = 56 7 * 9 = 63
8 * 1 = 8 8 * 2 = 16 8 * 3 = 24 8 * 4 = 32 8 * 5 = 40 8 * 6 = 48 8 * 7 = 56 8 * 8 = 64 8 * 9 = 72
9 * 1 = 9 9 * 2 = 18 9 * 3 = 27 9 * 4 = 36 9 * 5 = 45 9 * 6 = 54 9 * 7 = 63 9 * 8 = 72 9 * 9 = 81
请按任意键继续. . .
```

图 4.15　例 4-18 程序运行结果之二

```
for i in range(1, 10, 1):                       #控制行
    for j in range(1, i+1, 1):                  #控制列
        print i, '*', j, '=', i*j,
    print '\n'                                  #每行末尾的换行
```

程序运行结果如图 4.16 所示。

```
C:\WINDOWS\system32\cmd.exe - PAUSE
1 * 1 = 1

2 * 1 = 2 2 * 2 = 4

3 * 1 = 3 3 * 2 = 6 3 * 3 = 9

4 * 1 = 4 4 * 2 = 8 4 * 3 = 12 4 * 4 = 16

5 * 1 = 5 5 * 2 = 10 5 * 3 = 15 5 * 4 = 20 5 * 5 = 25

6 * 1 = 6 6 * 2 = 12 6 * 3 = 18 6 * 4 = 24 6 * 5 = 30 6 * 6 = 36

7 * 1 = 7 7 * 2 = 14 7 * 3 = 21 7 * 4 = 28 7 * 5 = 35 7 * 6 = 42 7 * 7 = 49

8 * 1 = 8 8 * 2 = 16 8 * 3 = 24 8 * 4 = 32 8 * 5 = 40 8 * 6 = 48 8 * 7 = 56 8 * 8 = 64

9 * 1 = 9 9 * 2 = 18 9 * 3 = 27 9 * 4 = 36 9 * 5 = 45 9 * 6 = 54 9 * 7 = 63 9 * 8 = 72 9 * 9 = 81

请按任意键继续. . .
```

图 4.16　例 4-18 程序运行结果之三

注意：多重循环的执行过程是，外层循环每执行一次，内层循环就要从头开始执行一轮，执行多次。多重循环总的循环次数等于每一层循环次数相乘。九九乘法的双重循环

中，外层循环变量 i 取 1 时，内层循环执行 9 次(J 依次取 1,2,…,9)，当外层循环变量 I＝2 时，内层循环同样要重新执行 9 次(J 再依次取 1,2,…,9)，故，总的循环次数为 9＊9＝81 次。

【例 4-19】 计算 1!＋2!＋3!＋…＋10!的值。

【解析】 此题与 1＋2＋3＋…＋10 极为相似，只是将 1 转变为 1!，2 转变为 2!，3 转变为 3!，…，10 转变为 10!。因此，外层循环为 10 项内容累加，内层循环为每个数的累积。

```
sum=0
for i in range(1,11):
    s=1
    for j in range(1,11):
        s=j*s
    sum=sum+s
print sum
```

【例 4-20】 编写程序，在窗体上输出的结果如图 4.17 所示。



图 4.17 例 4-19 程序运行结果之一

【解析】 构造二重循环，首先实现内层循环，作为步骤一，输出一行五个星号。其次，实现外层循环，即步骤二，将每行五个星号重复三边，即输出三行，每行五个星号。

步骤一：

```
for i in range(0, 5):
    print "* ",           #注意逗号
```

运行结果如图 4.18 所示。



图 4.18 例 4-19 程序运行结果之二

步骤二：

```
for i in range(0,3):
    for j in range(0, 5):
```

```
        print "* ",
    Print
```

如果输出如图 4.19 所示。



图 4.19　例 4-19 程序运行结果之三

【解析】　分析星号数(J)和行数(I)的关系，如表 4.2 所示。

表 4.2　空格数、星号数和行数的关系表

I(行数)	J(星号数)	I(行数)	J(星号数)
1	1	3	5
2	3		

推出公式：J＝2＊i－1

```
for i in range(1,4):
    for j in range(0, 2* i-1):
        print "* ",
    print
```

如果输出如图 4.20 所示。



图 4.20　例 4-19 程序运行结果之四

【解析】　分析空格数(K)、星号数(J)和行数(I)的关系，如表 4.3 所示。

表 4.3　空格数、星号数和行数的关系表

I(行数)	K(空格数)	J(星号数)
1	2	1
2	1	3
3	0	5

推出公式：J＝2＊i－1 和 k＝3－i

```
for i in range(1,4):
    for j in range(0, 3-i):
        print " ",
    for k in range(0,2*i-1):
        print "*",
    print
```

4.6 习题

1. 求 1～100 之间所有的素数，并统计素数的个数。

2. 求 200 以内能被 17 整除的最大正整数。

3. 设 m＝1＊2＊3＊…＊n，求 m 为不大于 20000 时最大的 n。

4. 勾股定理中 3 个数的关系是：$a^2+b^2=c^2$。编写一个程序，输出 30 以内满足上述条件的整数组合，如 3，4，5 就是一个组合。

5. 625 这个数字很特别，625 的平方等于 390625，刚好其末 3 位是 625 本身。除了 625，还有其他 3 位数有这个特征吗？请编写程序，寻找所有这样的 3 位数：它的平方的末 3 位是这个数字本身。

6. 求 1～100 之间能被 7 整除，但不能同时被 5 整除的所有整数。

7. 编写程序，输出如图 4.21 所示。

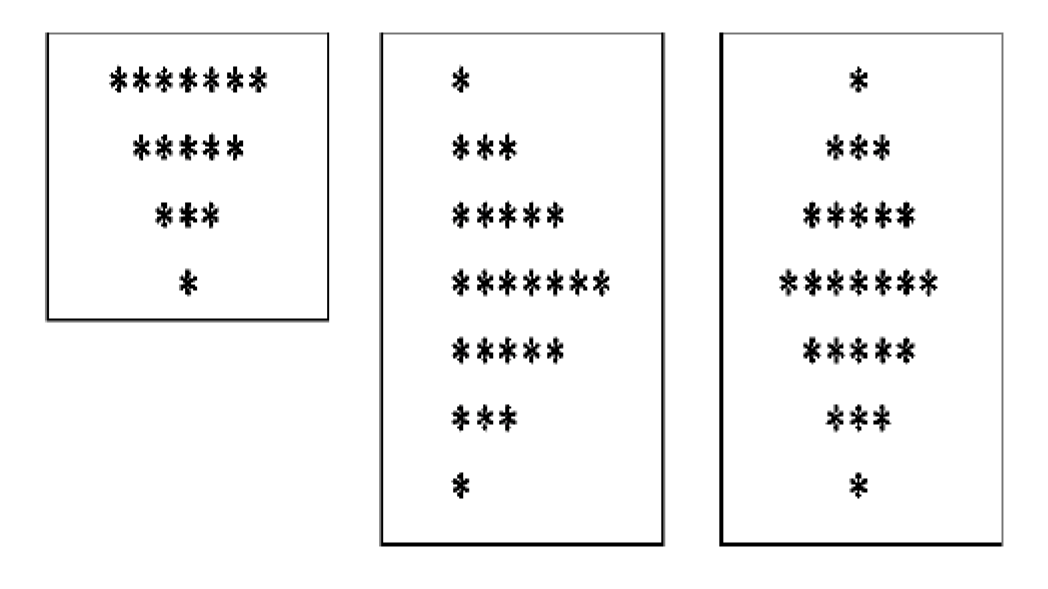
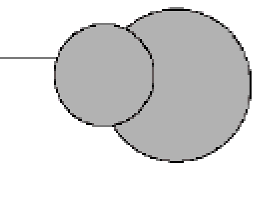


图 4.21　习题 4.6.7 程序运行结果

第5章



序列与字典

序列是程序设计中经常使用的数据存储方式，具有顺序编号的特征。本章介绍Python提供的几种序列——字符串、列表、元组，详细解释序列的通用操作方式和各自的特性及操作方式。最后介绍字典的概念和常用操作。

5.1 序列

5.1.1 序列概念

序列的成员有序排列，可以通过下标访问到一个或者几个成员，类似于C和Visual Basic语言中一维数组和多维数组等。Python提供字符串、列表、元组等序列类型。

5.1.2 序列通用操作

字符串、列表、元组等序列类型具有索引操作符和切片操作符。使用索引操作符可以从序列中得到特定元素。使用切片操作符能够获取序列的多个元素，即一部分序列。

(1) 索引

序列型的数据类型的每个元素都有一个标号用于标识其位置，从左至右依次是0，1，…，n，从右向左计数来存取元素称为负数索引，依次是－1，－2，…，－n。li[－n]==li[len(list)－n]。

【例5-1】 索引举例。

```
>>>l1=[1,1.3,"a"]
>>>l1[0]
1
>>>l1[-1]
'a'
```

注意：Python从0开始计数。

(2) 切片

序列切片是指使用序列序号对截取序列中的任何部分，从而得到一个新序列。切片操作符是在[ ]内提供一对可选数字，用“:”分隔。冒号前的数表示切片的开始位置，冒号后的数字表示切片截止(但不包含)位置。

【例 5-2】 切片举例。

```
>>>l1=[1,1.3,"a"]
>>>l1[1:2]          #取出位置从 1 开始到位置为 2 的字符,但不包含偏移为 2 的元素
[1.3]
>>>l1[:2]           #不指定第一个数,切片从第一个元素,直到(但不包含)偏移为 2 的元素
[1, 1.3]
>>>l1[1:]           #不指定第二个数,从偏移为 1 直到末尾之间的元素
[1.3, 'a']
>>>l1[:]            #数字都不指定,则返回整个列表的一个副本
[1, 1.3, 'a']
```

(3) 加

两个整数类型相加是整数值做加法,而对于两个序列,加法则表示连接操作,需要注意,进行操作的两个序列必须是相同的类型(字符串、列表、元组)才可以连接。

【例 5-3】 加法举例。

```
>>>l1=[1,1.3,"a"]
>>>l2=["d",['one','two']]
>>>l1+l2
[1, 1.3, 'a', 'd', ['one', 'two']]
```

(4) 乘

序列的乘法表示将原来的序列重复多次。

【例 5-4】 乘法举例。

```
>>>l1=[1,1.3,"a"]
>>>l1*3
[1, 1.3, 'a', 1, 1.3, 'a', 1, 1.3, 'a']
```

(5) 检查某个元素是否属于序列

判断某个元素是否在序列中,可以使用 in 和 not in 运算符。

【例 5-5】 in 举例。

```
>>>1.3 in l1
True
>>>2 in l1
False
>>>2 not in l1
True
```

序列除了通用操作之外,还具有如下一些函数。

(1) len(seq)

求出序列所包含的元素数量。

【例 5-6】 len()举例。

```
>>>l1=[1,5,9]
>>>len(l1)
```

```
3
```

（2） min(seq)

求出序列中最小值。

【例 5-7】　min()举例。

```
>>>l1=[1,5,9]
>>min(l1)
1
```

（3） max(seq)

求出序列中最大值。

【例 5-8】　max()举例。

```
>>>l1=[1,5,9]
>>>max(l1)
9
```

（4） sum(seq[index1,index2])

求出序列中切片之间的和，序列元素必须是数值。

【例 5-9】　sum()举例。

```
>>>l1=[1,5,9]
>>>sum(l1[0:3])
15
```

5.2　列表

5.2.1　列表概念

现实生活中的购物清单、手机通讯录等都可以看作一个列表，列表(list)是一组有序项目的数据结构。Python 创建列表时，解释器在内存中生成一个类似数组的数据结构存储数据，数据项自下而上存储，如图 5.1 所示。

4	True
3	[10, 20]
2	5
1	2.0
0	apple

图 5.1　列表存储方式

Python 列表可以包含混合类型的数据，即在一个列表中的数据类型可以各不相同。列表中的每一个数据称为元素，元素用逗号分隔并放在一对中括号“[”和“]”中。

列表举例：

```
[10, 20, 30, 40]                #所有元素都是整型数据的
列表
[' frog', 'cat', 'dog']         #所有元素都是字符串的列表
['apple', 2.0, 5, [10, 20],True]
```

＃列表中包含字符串、浮点类型、整型、列表类型和布尔类型。

5.2.2 列表操作

下面介绍列表操作。

(1) 创建列表：使用“＝”将一个列表赋值给变量。

例如：

```
>>>a_list=['a', 'b', 'c']
```

(2) 读取元素：用列表名加元素序号访问列表中某个元素。

例如：

```
>>>a_list=['a', 'b', 'c']
>>>print(a_list[2])
c
```

(3) 修改 list 中的元素：只须直接给元素赋值。

```
>>>a_list=['a', 'b', 'c']
>>>a_list[0]=123
>>>print a_list
[123, 'b', 'c']
```

(4) 增加元素。

方法一：使用“＋”将一个新列表附加在原列表的尾部。

```
>>>a_list=[1]
>>>a_list=a_list+['a', 2.0]
>>>a_list
[1, 'a', 2.0]
```

方法二：使用 append()方法向列表尾部添加一个新元素。

```
>>>a_list=[1, 'a', 2.0]
>>>a_list.append(True)
>>>a_list
[1, 'a', 2.0, True]
```

方法三：使用 extend()方法将一个列表添加在原列表的尾部。

```
>>>a_list=[1, 'a', 2.0, True]
>>>a_list.extend(['x', 4])
>>>a_list
[1, 'a', 2.0, True, 'x', 4]
```

方法四：使用 insert()方法将一个元素插入到列表的任意位置。

```
>>>a_list=[1, 'a', 2.0, True, 'x', 4]
>>>a_list.insert(0, 'x')
>>>a_list
```

```
['x', 1, 'a', 2.0, True, 'x', 4]
```

（5）检索元素。

使用 count()方法计算列表中某个元素出现的次数。

```
>>>a_list=['x', 1, 'a', 2.0, True, 'x', 4]
>>>a_list.count('x')
2
```

使用 in 运算符返回某个元素是否在该列表中。

```
>>>a_list=['x', 1, 'a', 2.0, True, 'x', 4]
>>>3 in a_list
False
>>>2.0 in a_list
True
```

（6）删除元素。

方法一：使用 del 语句删除某个特定位置的元素。

```
>>>a_list=['x', 1, 'a', 2.0, True, 'x', 4]
>>>del a_list[1]
>>>a_list
['x', 'a', 2.0, True, 'x', 4]
```

方法二：使用 remove 方法删除某个特定值的元素。

```
>>>a_list=['x', 'a', 2.0, True, 'x', 4]
>>>a_list.remove('x')
>>>a_list
['a', 2.0, True, 'x', 4]
>>>a_list.remove('x')
>>>a_list
['a', 2.0, True, 4]
>>>a_list.remove('x')
Traceback (most recent call last):
    File "<pyshell#51>", line 1, in<module>
        a_list.remove('x')
ValueError: list.remove(x): x not in list
```

方法三：使用 pop(参数)方法弹出指定位置的元素，默认参数时弹出最后一个元素。

```
>>>a_list=['a', 2.0, True, 4]
>>>a_list.pop()          #默认参数时弹出最后一个元素
4
>>>a_list
['a', 2.0, True]
>>>a_list.pop(1)
```

```
2.0
>>>a_list
['a', True]
>>>a_list.pop(1)
True
>>>a_list
['a']
>>>a_list.pop()
'a'
>>>a_list
[ ]
>>>a_list.pop()
Traceback (most recent call last):
    File "<pyshell#61>", line 1, in<module>
        a_list.pop()
IndexError: pop from empty list
```

列表方法如表 5.1 所示。

表 5.1　列表方法

函　　数	描　　述
alist.append(obj)	列表末尾增加元素 obj
alist.count(obj)	统计元素 obj 出现次数
alist.extend(sequence)	用 sequence 扩展列表
alist.index(obj)	返回列表中元素 obj 的索引
alist.insert(index,obj)	在 index 索引之前添加元素 obj
alist.pop(index)	删除索引的元素
alist.remove(obj)	删除指定元素
alist.reverse()	原地反转列表元素顺序
alist.sort(obj)	为列表排序

【例 5-10】 成绩统计问题。

【问题描述】 随意输入 10 个百分制成绩，输出优(100～90)、良(89～80)、中(79～60)、及格(69～60)、差(59～0) 5 个等级的人数。

【解析】 输入的成绩存放在列表中，设置 a,b,c,d,e 5 个变量对应优、良、中、及格、差 5 个等级，对每个成绩进行分析判断，属于哪个等级范围就将对应变量值加 1。程序流程图如图 5.2 所示。

```
score= [68, 75, 32, 99, 78, 45, 88, 72, 83, 78]
a= 0
b= 0
```

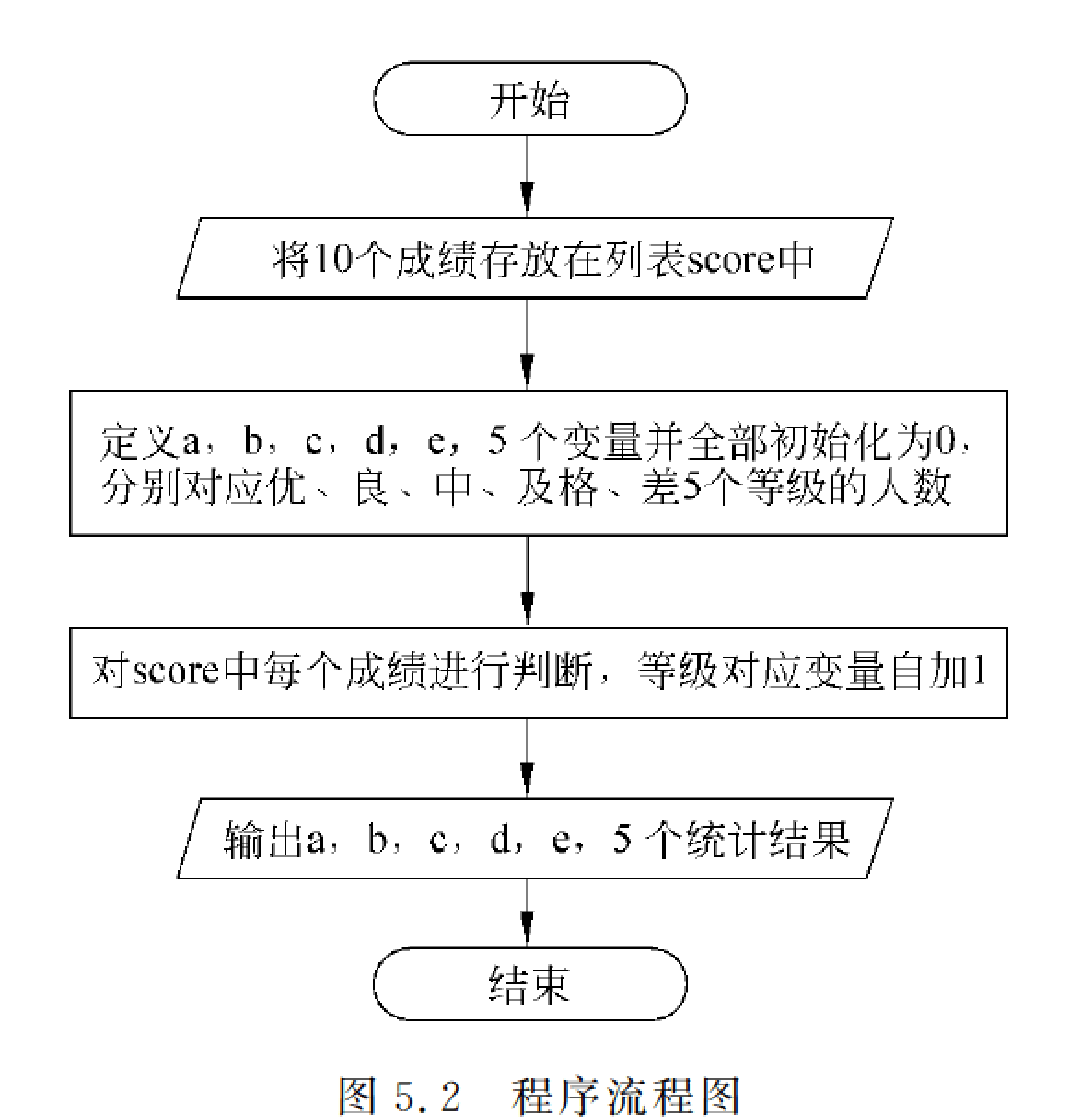


图 5.2　程序流程图

```
c=0
d=0
e=0
#输出所有成绩
print "成绩分别为：",
for s in score:
    print s,
#换行
print
#对成绩进行分段统计
for s in score:
    if s<60:
        e=e+1
    elif s<70;
    d=d+1
    elif s<80:
        c=c+1
    elif s<90:
        b=b+1
    else:
        a=a+1
print "分段统计结果：优",a,"人,良",b,"人,中",c,"人,及格",d,"人,差" ,e,"人"
```

5.3　元组

5.3.1　元组概念

元组(Tuple)和列表类似,但其元素是不可变,即元组一旦创建,用任何方法都不可

以修改其元素，元组相当于只读列表。

元组与列表相比，有如下相同点：

(1) 元组的元素与列表一样按定义的次序进行排序。

(2) 元组的负数索引与列表一样，从元组的尾部开始计数。

(3) 元组与列表一样也可以使用分片。

元组与列表相比，有如下相异点：

(1) 定义元组时所有元素是放在一对圆括号"("和")"中，而不是方括号。

(2) 不能向元组增加元素，元组没有 append 或 extend 方法。

(3) 不能从元组删除元素，元组没有 remove 或 pop 方法。

(4) 元组没有 index 方法。但是可以使用 in 方法。

(5) 元组可以在字典中被用作"键"，但是列表不行。

元组适于只能进行遍历操作的运算，由于对数据进行"写保护"操作，使得代码安全，操作速度比列表快。

5.3.2 元组操作

下面介绍元组操作。

(1) 访问元组

元组可以使用下标索引来访问元组中的值。

```
>>>tup1= ('a', 'b', 1997, 2000);
>>>tup2= (1, 2, 3, 4, 5, 6, 7);
>>>print "tup1[0]: ", tup1[0]
>>>print "tup2[1:5]: ", tup2[1:5]
tup1[0]:  a
tup2[1:5]:  [2, 3, 4, 5]
```

(2) 元组连接

元组中的元素值是不允许修改的，但可以对元组进行连接组合。

```
>>>tup1= (12, 34.56);
>>>tup2= ('abc', 'xyz');
#tup1[0]=100                 #修改元组元素操作是非法的
>>>tup3=tup1+tup2;           #创建一个新的元组
>>>print tup3;
(12, 34.56, 'abc', 'xyz')
```

(3) 删除元组

元组中的元素值是不允许删除的，但可以使用 del 语句删除整个元组。

```
>>>tup= ('physics', 'chemistry', 1997, 2000);
>>>del tup;
>>>print tup;
Traceback (most recent call last):
```

```
    File "<pyshell#21>", line 1, in<module>
        print tup
NameError: name 'tup' is not defined
```

元组方法如表 5.2 所示。

表 5.2 元组方法

元组表达式	描 述	结 果
len((1,2,3))	计算元素个数	3
(1,2,3)+(4,5,6)	连接	(1,2,3,4,5,6)
('Hi! ')*4	复制	('Hi! ','Hi! ','Hi! ','Hi! ')
3in(1,2,3)	元素是否存在	True

5.4 字符串

5.4.1 字符串操作

和其他高级语言一样,Python 也提供字符串方法用于字符串的操作。

(1) index 举例

```
>>> s="Python"
>>> s.index('P')
0
>>> s.index('h',1,4)
3
>>> s.index('y',3,4)
Traceback (most recent call last):
    File "<pyshell#27>", line 1, in<module>
        s.index('y',3,4)
ValueError: substring not found
>>> s.index('h',3,4)
3
```

(2) find 举例

```
>>> s="Python"
>>> s.find('s')
-1
>>> s.find('t',1)
2
```

(3) replace 举例

```
>>> s="Python"
```

```
>>>s.replace('h','i')
'Pytion'
```

(4) count 举例

```
>>>s="Python"
>>>s.count('n')
1
```

(5) split 举例

split 默认以字符串中的空格作为分隔符进行分割,分割后的每一段都是一个新的字符串,最终返回这些字符串组成一个字符串,原来字符串中的空格不再存在。

```
>>>s="Python"
>>>s.split()
['Python']
>>>s="hello Python i like it"
>>>s.split()              #默认以空格作为分隔符
['hello', 'Python', 'i', 'like', 'it']
>>>s='name:haha,age:20|name:python,age:30|name:fef,age:55'
>>>print s.split('|')     #以'|'作为分隔符
['name:haha,age:20','name:python,age:30','name:fef,age:55']
>>>x,y=s.split('|',1)
>>>print x
name:haha,age:20
>>>print y
name:python,age:30|name:fef,age:55
```

(6) join 举例

```
>>>li=['apple','peach','banana','pear']
>>>sep=','
>>>s=sep.join(li)
>>>s
'apple,peach,banana,pear'
```

5.4.2 字符串、列表、元组转换

列表、元组和字符串之间可以互相转换,如下所示。

(1) 字符串转换为列表,使用 List()

```
>>>str="123,45"
>>>list(str)
['1', '2', '3', ',', '4', '5']
```

(2) 字符串转换为元组,使用 tuple()

```
>>>str="123,45"
```

```
>>> tuple(str)
('1', '2', '3', ',', '4', '5')
```

(3) 列表和元组转换为字符串,必须使用 join 方法

```
>>> s= ('a', 'b', 'c', 'd')          #元组
>>> "".join(tuple(s))
'abcd'
>>> s= ['a', 'b', 'c', 'd']          #列表
>>> "".join(list(s))
'abcd'
```

字符串方法如表 5.3 所示。

表 5.3 字符串方法

函　　数	描　　述
s. index (sub,[start,end])	定位子串 sub 在 s 里第一次出现的位置
s. find(sub,[start,end])	与 index 函数一样,但如果找不到会返回－1
s. replace(old,new [,count])	替换 s 里所有 old 子串为 new 子串,count 指定多少个可被替换
s. count(sub[,start,end])	统计 s 里有多少个 sub 子串
s. split()	字符串的 split 函数默认分隔符是空格‘ ’,如果没有分隔符,就把整个字符串作为列表的一个元素
s. join()	join()方法是 split()方法的逆方法,用来把字符串连接起来
s. lower()	返回将大写字母变成小写字母的字符串
s. upper()	返回将小写字母变成大写字母的字符串

5.5 字典

5.5.1 字典概念

【例 5-11】 根据学生的姓名查找其对应的成绩。

【解析】 若采用列表实现,则需要 names 和 scores 两个列表,列表中元素的次序一一对应,例如 zhou→95,Bob→75 等,如下所示:

```
names= ['zhou', 'Bob', 'Tracy']
scores= [95, 75, 85]
```

通过名字查找对应的成绩,先在 names 中遍历找到所需查找的名字,再从 scores 遍历取出对应的成绩,如果列表越长,则查找耗时越长。为了解决这个问题,Python 提供了字典(dict)。字典在其他程序设计语言中称为映射(map),通过键值对(key-value)存储数据,具有极快的查找速度。

采用字典实现例 5-11,只需创建“名字”-“成绩”的键值对,便可直接通过名字查找成

绩。字典实现代码如下：

```
>>>d={'zhou': 95, 'Bob': 75, 'Tracy': 85}
>>>d['zhou']
95
```

字典通过用空间来换取时间，与列表比较，有以下几个特点：

（1）字典查找和插入的速度极快，不会随着“键”的增加而增加。

（2）字典需要占用大量的内存。

（3）字典是无序的对象集合，字典中的值通过键来存取，而不是通过偏移存取。

字典由一对键和值构成，键和值之间用冒号间隔，元素项之间用逗号间隔，整体用一对大括号“{”和“}”括起来。字典语法结构如下所示：

```
dict_name={key:value,key:value}
```

字典有如下特性：

（1）字典的值可以是任意数据类型，包括字符串、整数、对象，甚至字典。

（2）不允许同一个键重复出现，如果同一个键被赋值两次，后一个值会覆盖前面的值。

```
>>>dict={'Name': 'Zara', 'Age': 7, 'Name': 'Zhou'}
dict['Name']:  Zhou
```

（3）键必须不可变，只能由数、字符串或元组充当，不能用列表。

```
>>>dict={['Name']: 'Zara', 'Age': 7}
Traceback (most recent call last):
    File "<pyshell#0>", line 1, in<module>
        dict={['Name']: 'Zara', 'Age': 7}
TypeError: unhashable type: 'list'
```

5.5.2 字典操作

下面介绍字典操作。

（1）字典元素的访问

① keys()方法返回一个包含所有键的列表。

```
>>>dict={'zhou': 95, 'Bob': 75, 'Tracy': 85}
>>>dict.keys()
['Bob', 'Tracy', 'zhou']
```

② has_key()方法检查字典中是否存在某一键。

```
>>>dict={'zhou': 95, 'Bob': 75, 'Tracy': 85}
>>>dict.has_key('zhou')
True
```

③ values()方法返回一个包含所有值的列表。

```
>>>dict={'zhou': 95, 'Bob': 75, 'Tracy': 85}
>>>dict.values()
[75, 85, 95]
```

④ get()方法根据键返回值，如果不存在输入的键，返回 None。

```
>>>dict={'zhou': 95, 'Bob': 75, 'Tracy': 85}
>>>dict.get('Bob')
75
```

⑤ items()方法返回一个由(key,value)组成的元组。

```
>>>dict={'zhou': 95, 'Bob': 75, 'Tracy': 85}
>>>dict.items()
[('Bob', 75), ('Tracy', 85), ('zhou', 95)]
```

(2) 字典元素的删除

① del()方法允许使用键从字典中删除元素。

```
>>>dict={'zhou': 95, 'Bob': 75, 'Tracy': 85}
>>>del dict['zhou']
>>>print dict
{'Bob': 75, 'Tracy': 85}
```

② clear 方法清除字典中所有元素。

```
>>>dict={'zhou': 95, 'Bob': 75, 'Tracy': 85}
>>>dict.clear()
>>>dict
{ }
```

注意：空的大括号集合表示没有元素的字典。

③ pop 方法删除一个关键字并返回它的值。

```
>>>dict={'zhou': 95, 'Bob': 75, 'Tracy': 85}
>>>dict.pop('zhou')
95
>>>print dict
{'Bob': 75, 'Tracy': 85}
```

(3) update 方法

update 方法类似于合并，把一个字典的键和值合并到另一个字典，覆盖相同键的值。

```
>>>tel={'gree': 4127, 'mark': 4127, 'jack': 4098}
>>>tel1={'gree':5127, 'pang':6008}
>>>tel.update(tel1)
>>>tel
```

```
{'gree': 5127, 'pang': 6008, 'jack': 4098, 'mark': 4127}
```

(4) in 运算

字典里的 in 运算用于判断某键是否在字典里,对于 value 值不适用。

```
>>>tel1={'gree':5127, 'pang':6008}
>>>'gree'in tel1
True
```

字典方法如表 5.4 所示。

表 5.4 字典方法

函　数	描　述
aDic.clear()	删除字典所有元素
aDic.copy()	返回字典副本
aDic.get(key)	返回字典的 key
aDic.has_key(key)	检查字典是否有给定的键
aDic.items()	返回表示字典(键、值)对应表
aDic.keys()	返回字典键的列表
aDic.pop(key)	删除并返回给定键
aDic.values()	返回字典值的列表

【例 5-12】 成绩排序问题。

【问题描述】 已知 10 个学生的姓名和成绩,请找出其中的最高分和最低分,并求出 10 个学生的平均分。

【解析】 将学生姓名和成绩以键值对形式存放在字典中。初始化存放最高分的变量为 0,存放最低分的变量为 100。取出字典中每个键(姓名)值(成绩)对,其值与目前的最高分比较,若大于最高分,则更新最高分;与目前的最低分比较,若小于最低分,则更新最低分。最后输出平均分,如图 5.3 所示。

```
studscore={"周哲": 45, "周恒": 78, "孟君": 40, "庞胜利": 96, "张莉": 65, "王小银": 90, "王江舟": 78, "师沫迪": 99, "张利": 60, "李晓戈": 87}
maxscore=0                    #最高分的变量 maxscore
maxstudname=' '
minscore=100                  #最低分的变量 minscore
minstudname=' '
avrscore=0
studnum=len(studscore)
#输出所有成绩:
print "成绩分别为: "
for key in studscore.keys():
        print key,studscore[key],";",
#换行
print
```

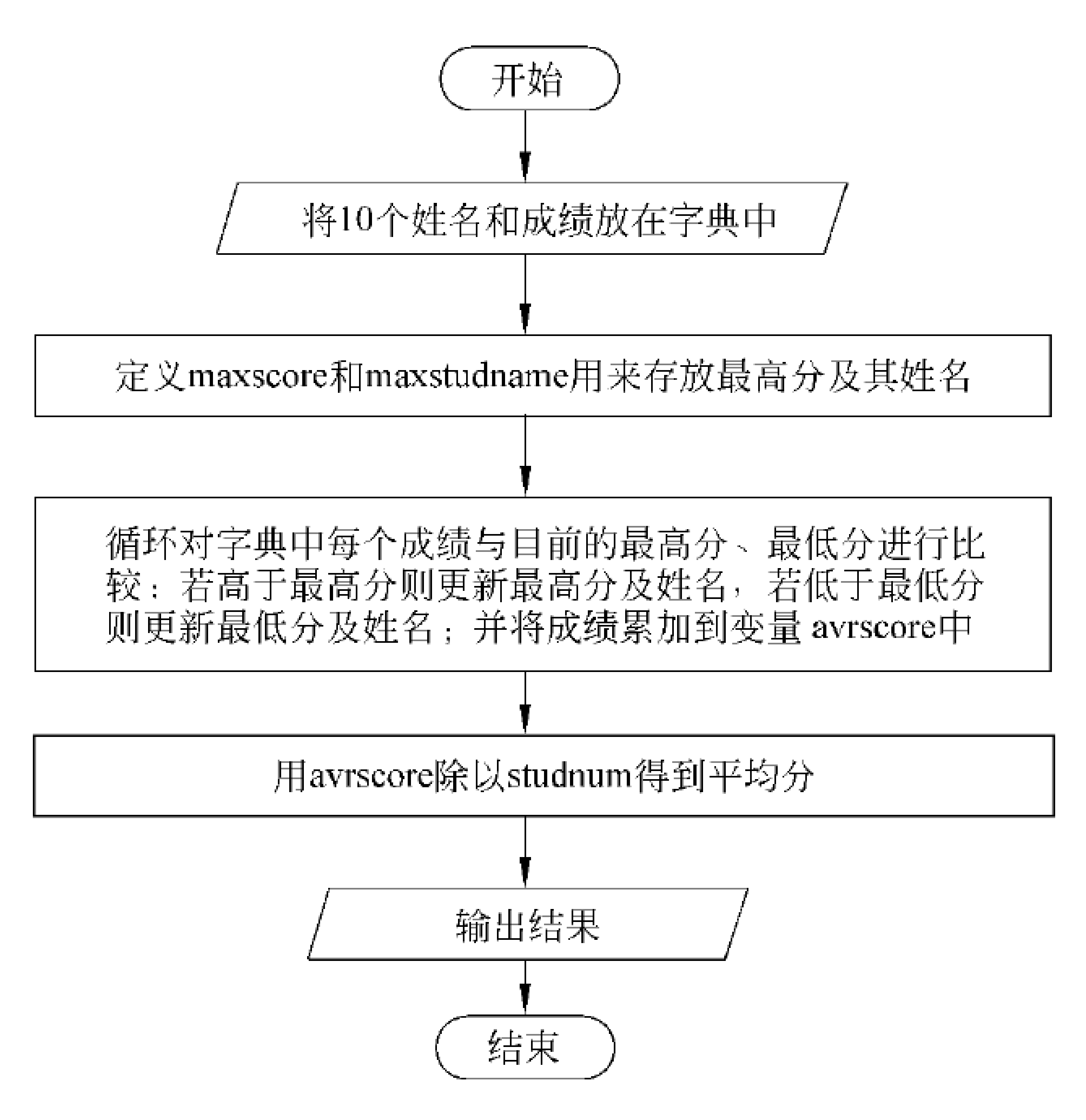


图 5.3 例 5-12 流程图

```
#进行成绩统计
for key in studscore.keys():
    if studscore[key]>maxscore:
        maxscore=studscore[key]
        maxstudname=key
    if studscore[key]<minscore:
        minscore=studscore[key]
        minstudname=key
    avrscore=avrscore+studscore[key]
avrscore=avrscore/studnum
print "全班共有",studnum,"人,平均成绩为：",avrscore,"分。"
print "最高分是：",maxstudname,maxscore,"分"
print "最低分是：",minstudname,minscore,"分"
```

5.6 习题

1. 输入一段英文文章,求其长度,并求出包含多少个单词。

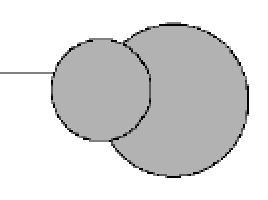
2. 输入 10 个成绩,进行优、良、中、及格和不及格的统计。

3. 输入 10 个学生的姓名和成绩构成的字典,按照成绩大小排序。

4. 输入 10 个学生的姓名和年龄构成的字典,读出其键和值,并分别保存输出到两个列表中。

5. 任意输入一串字符,输出其中不同字符以及各自的个数。例如,输入“abcdefgabc”,输出为 a→2,b→2,c→2,d→1,e→1,f→1,g→1。

第6章



数据结构与算法

本章介绍数据结构和算法相关内容。数据结构包括线性结构和非线性结构等内容，并从数据结构角度分析 Python 的序列结构，讲述算法的五个特性和三个层次，介绍有特点的数、经典趣味题等内容。

6.1 数据结构

著名计算机科学家沃思提出了一个公式：程序＝数据结构＋算法。其中，算法解决“如何操作数据”的问题。数据结构解决“如何描述数据”的问题，指定数据的类型和数据的组织形式。数据结构在计算机学科中具有核心地位，如图 6.1 所示。

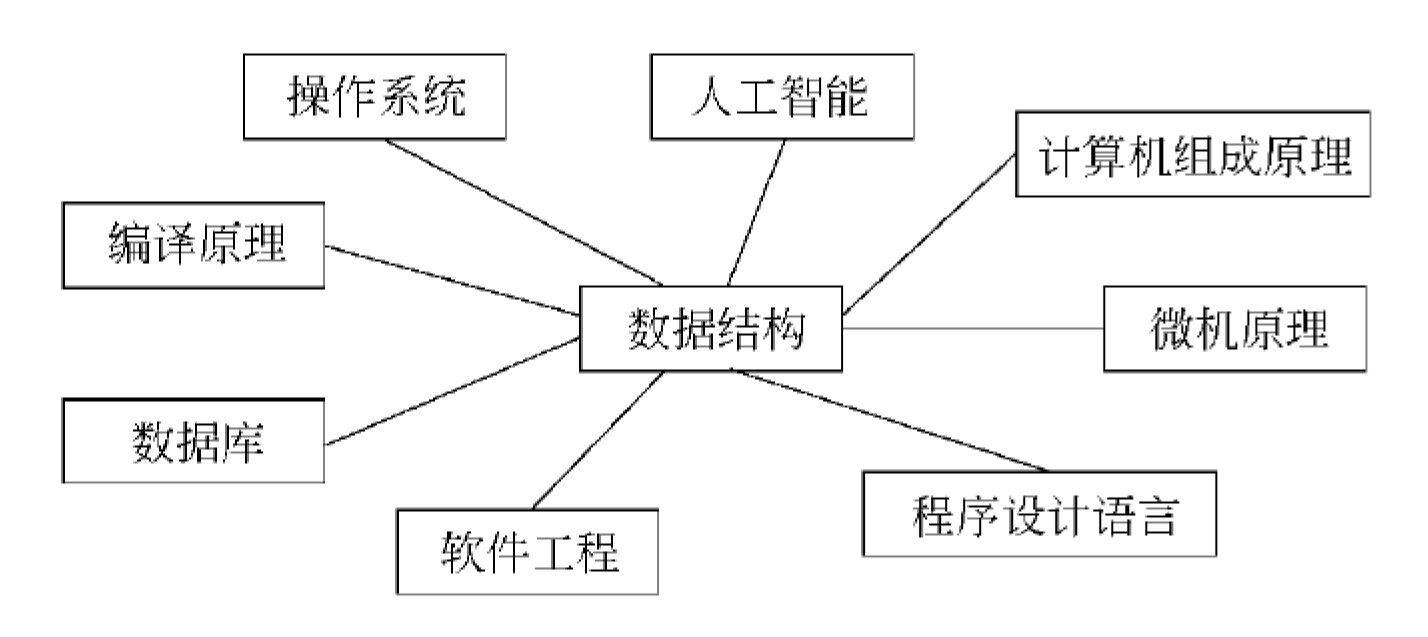


图 6.1 数据结构在计算机学科中的地位

数据结构研究相关的各种信息如何表示、组织和存储与加工处理，“数据结构”中数据的“关系”指逻辑关系，与数据的物理存储无关。数据结构一般有线性结构和非线性结构。

6.1.1 线性结构

线性结构是指元素与元素之间是一对一的关系，一般有线性表、栈和队列等结构。

(1) 线性表

线性表(a_0，a_1，…，a_{n-1}，a_n)($n>0$)如图 6.2 所示，具有如下特点：

① 存在唯一的“第一元素”a_0；

② 存在唯一的一个“最后元素”a_n；

③ 除最后元素 a_n 在外，均有唯一的后继；

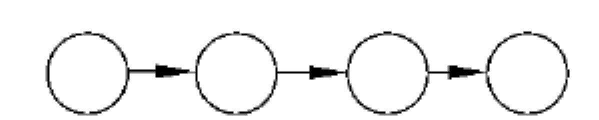


图 6.2 线性数据结构

④ 除第一元素 a_0 之外,均有唯一的前驱。

(2) 栈和队列

从数据结构角度讲,栈与队列也是线性表,不同之处在于其操作的特殊性(栈为 LIFO,队列为 FIFO),为操作受限的线性表。

栈是限定仅在表尾进行插入或删除操作的线性表,具有后进先出的特性,即最先进入的元素最后一个被释放,栈的逻辑结构如图 6.3 所示。

栈具有如下两个操作:

① 入栈(PUSH):最先插入的元素放在栈的底部。

② 出栈(POP):最后插入的元素最先出栈。

队列只允许在线性表的一端(队尾)进行插入(入队列),而在另一端(队头)进行删除(出队列)。队列的逻辑结构如图 6.4 所示。

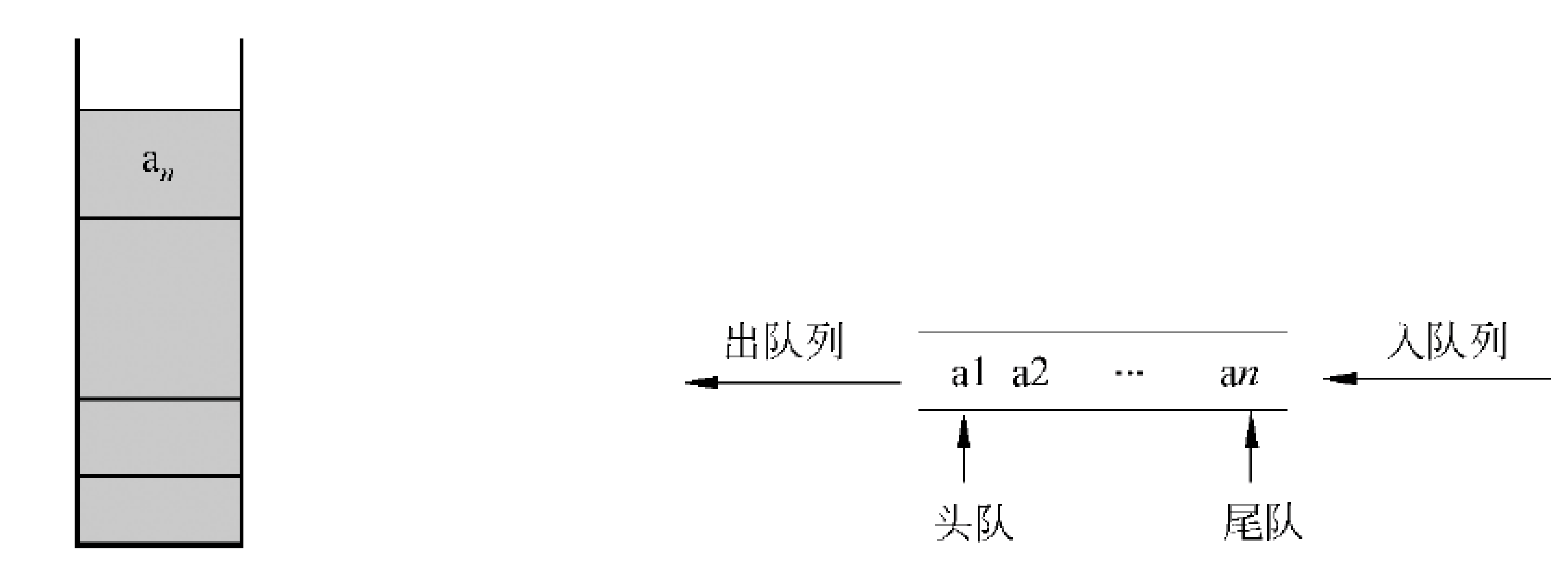


图 6.3 栈的逻辑结构　　图 6.4 队列的逻辑结构

6.1.2 非线性结构

非线性结构是指至少存在一个数据元素有两个或两个以上的直接后继(或直接前驱)元素的数据结构。非线性结构一般有树和图等结构。

(1) 树

树形结构用于描述一对多的关系,适合描述层次结构,如,人类的族谱等。树的逻辑结构如图 6.5 所示。

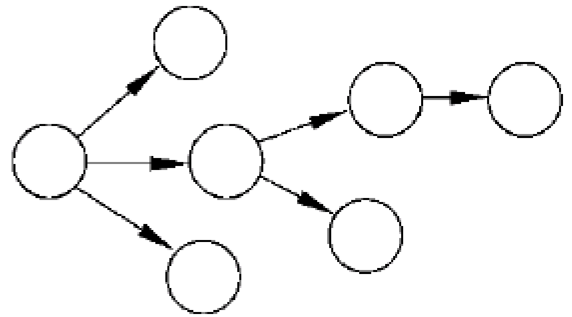


图 6.5 树的逻辑结构

其中,二叉树是树中最重要的概念。二叉树或为空树,或是由一个根结点加上两棵分别称为左子树和右子树的、互不相交的二叉树组成。二叉树的逻辑结构如图 6.6 所示。

(2) 图

图在各个领域都有着广泛的应用,如交通路线等,是一种比线性表和树更为复杂的数据结构。在线性表中,数据元素之间仅有线性关系,即每个数据元素只有一个直接前驱和一个直接后继;在树形结构中,数据元素之间有着明显的层次关系,虽然每一层上的数据元素可能和下一层中多个元素(孩子)相关,但只能和上一层中一个元素(双亲)相关;而在图形结构中,结点之间的关系可以是任意的,任意两个数据元素之间都可能相关,图适于描述多对多的关系,图 G 由两个集合 V 和 E 组成,记为

$$G=(V,E)$$

其中,V 是顶点的有穷非空集合,E 是 V 中顶点偶对(称为边)的有穷集。图的逻辑结构如图 6.7 所示。

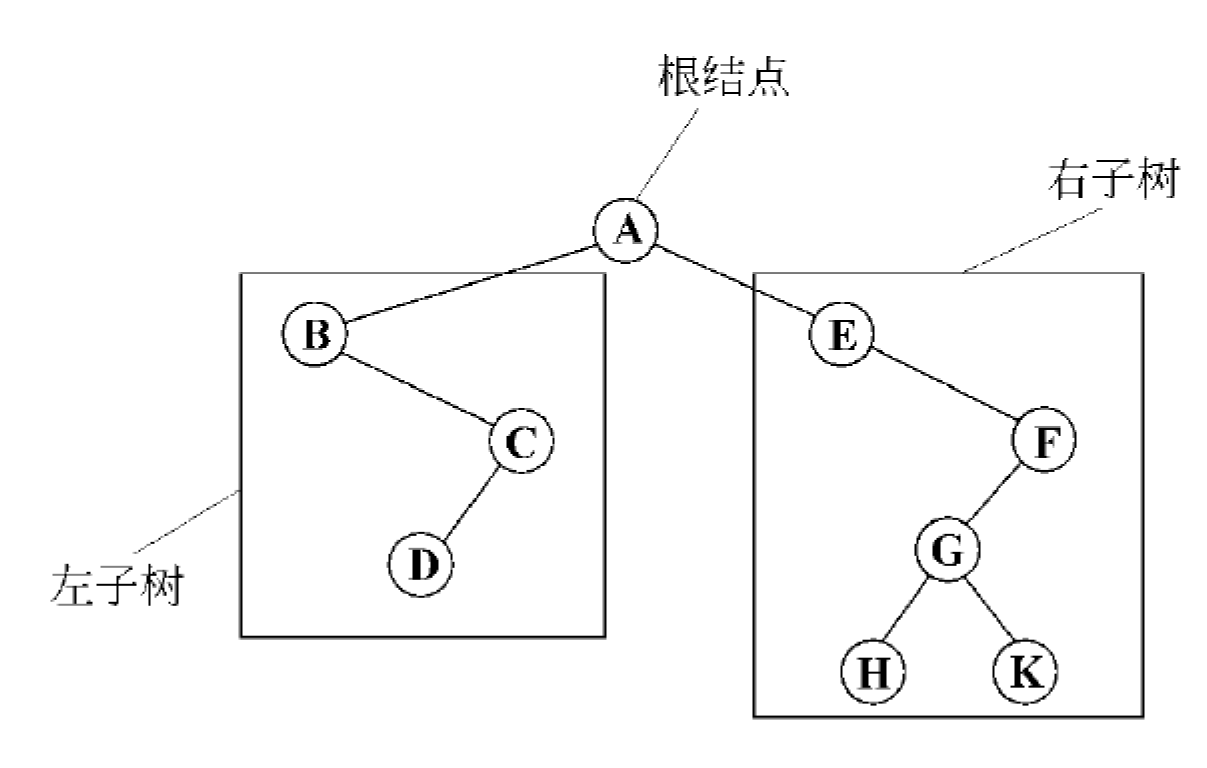


图 6.6 二叉树的逻辑结构

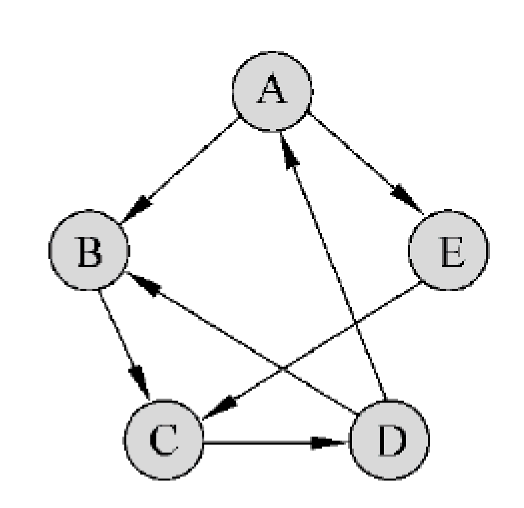


图 6.7 图的逻辑结构

6.1.3 序列与数据结构

Python 提供序列应用于数据结构的栈和队列。例如,序列中的列表可以快速实现栈,具体操作如下:

- list.append()方法对应入栈操作(Push);
- list.pop()方法对应出栈操作(Pop)。

列表也可以实现队列,但是不适合,因为从列表头部移去一个元素,列表中的所有元素都需要移动位置,效率较为低下。

【例 6-1】 序列实现栈结构。

```
>>> stack=list()
>>> stack.append('Apple');stack.append('banbana');stack.append('orange');
>>> print stack
['Apple', 'banbana', 'orange']
>>> stack.pop()
'orange'
>>> print stack
['Apple', 'banbana']
>>> stack.pop()
'banbana'
>>> print stack
['Apple']
```

6.2 查找和排序

6.2.1 查找

根据给定的某个值,在查找表中确定是否存在与给定值相等的数据元素(记录)。若

存在，则称“查找成功”，给出该值在查找表中的位置；否则称“查找不成功”。

基于线性表的查找有顺序查找、折半查找和分块查找等方法。

(1) 顺序查找

顺序查找又称线性查找，是最基本的查找方法之一。其查找方法为：从表的一端开始，向另一端逐个按给定值 kx 与关键码进行比较，若找到，则查找成功，并给出数据元素在表中的位置；若整个表检测完，仍未找到与 kx 相同的关键码，则查找失败，给出失败信息。

(2) 折半查找

顺序查找的算法简单，但平均查找长度较大，不适用于较长的查找表。若查找表中的数据元素是有序的，即升序或降序，则查找过程可以基于“折半”进行。

折半查找步骤如下所示：

首先，在序列元素 A(1)，A(2)，…，A(n) 中找到位置居中的 A(k)，即 k＝n/2，用 A(k) 将数组分成以下三个有序的序列：

第 1 序列：A(1)，A(2)，…，A(k－1)

第 2 序列：A(k)

第 3 序列：A(k＋1)，A(k＋2)，…，A(n)

然后，用 A(k) 与所要查找的数值 x 比较，若 x＝A(k)，则查找结束；若 x＜A(k)，则用同样的方法把序列 A(1)，A(2)，…，A(k－1) 分成三个序列；若 x＞A(k)，则也用同样的方法把序列 A(k＋1)，A(k＋2)，…，A(n) 分成三个序列，直到找到 x 或得到“x 找不到”的结论为止。

(3) 分块查找

分块查找的基本思想如下所示：

① 把线性表分成若干块，每块包含若干个记录，在每一块中记录的存放是任意的，但块与块之间必须排序，即分块有序。

② 建立一个索引表，把每块中的最大关键字值及每块的第一个记录在表中的位置和最后一个记录在表中的位置存放在索引项中。

分块查找的示意图如图 6.8 所示。

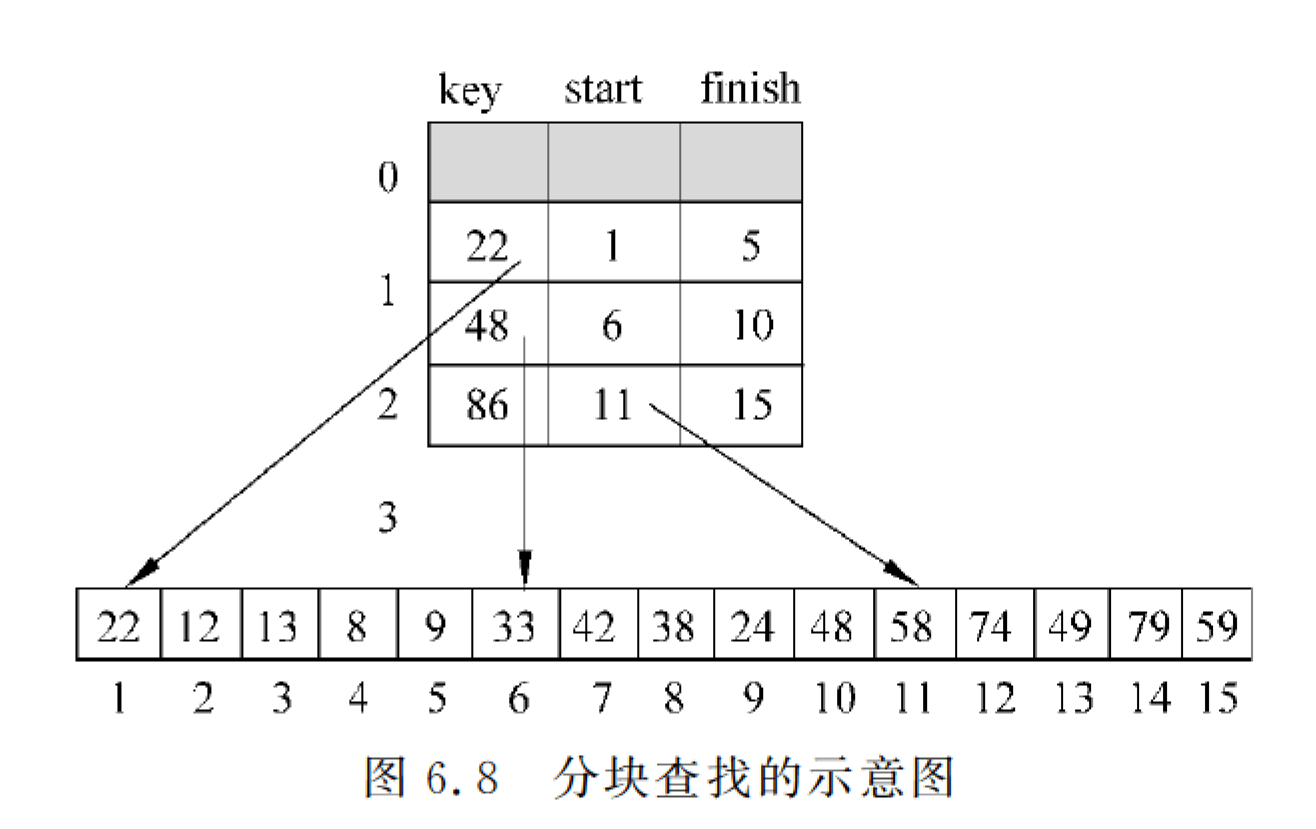


图 6.8 分块查找的示意图

索引顺序表的查找过程：

① 由索引确定记录所在区间(块)；

② 在某个区间(块)内进行查找。

可见，分块查找的过程也是一个“缩小区间”的查找过程。

6.2.2 排序

排序是将一组“无序”的记录序列调整为“有序”的记录序列。例如，将关键字序列(52,49,80,36,14,58,61,23,97,75)调整为(14,23,36,49,52,58,61,75,80,97)。排序的过程是一个逐步扩大有序序列区的过程，如图6.9所示。

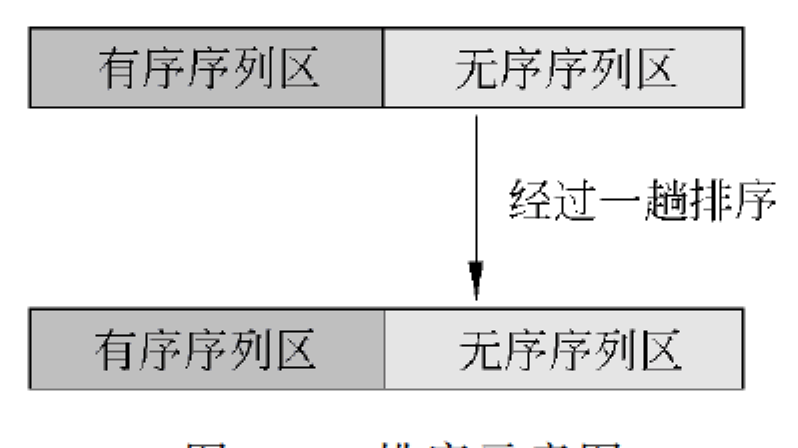


图6.9 排序示意图

排序方法一般有插入类、交换类和选择类等。

(1) 插入类

将无序子序列中的一个或几个记录“插入”到有序序列中，从而增加记录的有序序列的长度。

- 基于顺序查找的插入类排序称为直接插入排序。
- 基于折半查找的插入类排序称为折半插入排序。
- 基于逐趟缩小增量的插入类排序称为希尔排序。

(2) 交换类

通过“交换”无序序列中的记录从而得到其中关键字最小或最大的记录，并将它加入到有序子序列中，以此方法增加记录的有序子序列的长度。

交换类排序分为冒泡排序、一趟快速排序、快速排序等。

冒泡排序法步骤如下所示：

步骤1：将待排序的数组元素看成竖着排列的“气泡”，最小的元素为最小的“气泡”，较小的元素为较小的“气泡”。每一遍处理，就是自底向上检查一遍“气泡”序列，并时刻注意两个相邻的元素的顺序是否正确。如果发现两个相邻“气泡”的顺序不对，例如轻的“气泡”在重的“气泡”的下面，则交换它们的位置。一遍处理之后，“最轻”的“气泡”就浮到了最高位置。

步骤2：通过第二遍的处理之后，“次轻”的“气泡”就浮到了次高位置。

步骤3：如此反复，处理n－1次后，就可以完成“气泡”的有序排列。

(3) 选择类

从记录的无序子序列中“选择”关键字最小或最大的记录，并将它加入到有序子序列中，以此方法增加记录的有序子序列的长度。

选择类排序分为简单选择排序、树形选择排序和堆排序。

简单选择排序法步骤如下：

步骤1：从无序的数组中选择出最小的数，将这个最小数与数组的第1个位置上的元素进行交换，使得第1个数组位置上的数组元素为最小；

步骤2：除数组的第1个位置上的元素外，从剩下n－1个数组元素中，按“步骤1”选

出剩下 n−1 个数组元素中的最小值，将其与数组中的第 2 个位置上的元素交换，从而使得第 2 个位置上的数组元素为次小；

步骤 3：除第 1、第 2 个位置上的数组元素外，再从剩下的 n−2 个元素中选出最小的数，与数组中的第 3 个位置上的数组元素交换；

步骤 4：如此反复以上步骤，最后完成排序，成为递增序列。

6.3　算法

6.3.1　五个特性

算法（Algorithm）是通过对一定规范的输入进行处理，在有限时间内获得所要求的输出的整个过程，算法与具体的程序语言无关，一般具备以下五个特性：

(1) 确定性。算法的每个步骤都应确切无误，没有歧义性。

(2) 可行性。算法的每个步骤都必须满足计算机语言能够有效执行、可以实现的要求，并可得到确定的结果。

(3) 有穷性。算法包含的步骤必须是有限的，并在一个合理的时间限度内可以执行完毕，不能无休止地执行下去。例如计算圆周率，只能精确到某一位。

(4) 输入性。可以有多个输入，也可以没有输入（0 个输入）。

(5) 输出性。算法的目的是用于解决问题，必然要提供 1 个或多个输出。

【例 6-2】 从键盘上输入三角形的三个边，求三角形面积。

【解析】 其算法步骤如下所示：

步骤 1：从键盘上任意输入三个整数，用 a，b，c 存储。

步骤 2：判断 a，b，c 是否符合三角形的定义，即两边之和大于第三边。

步骤 3：如果符合，调用海伦公式，求出三角形面积 area。

步骤 4：输出 area。

下面，用算法的五个特性来分析例 6-2。

(1) 确定性。每一个步骤都有确定的含义，没有二义性。

(2) 可行性。每个步骤都可以用 Python 语句实现。

(3) 有穷性。只有 4 个步骤，是有限的。

(4) 输入性。有三个输入，a、b、c 分别代表三角形的三个边。

(5) 输出性。有一个输出，area 代表三角形的面积。

6.3.2　三个层次

Python 语言的学习大致分为以下两个方面：

(1) Python 语言本身的语法以及编程环境的掌握；

(2) 算法的学习。

算法学习可以分为如下三个层次，如表 6.1 所示。

表 6.1 算法的三个层次

层 次	内 容
第一层	一些基本的算法,如排序、查找、递归法等算法
第二层	涉及算法的时间复杂度和空间复杂度,如分治法、贪心算法、动态规划法等
第三层	涉及智能优化算法的学习,如遗传算法、蚁群算法、聚类算法等方法

第一层次是"算法基础教学阶段",学习基本的算法和程序设计方法,如查找、排序、递归程序设计等。典型的课程是"数据结构"。

第二层次是"算法提高教学阶段",学习一些重要的算法设计方法,如分治法、动态规划法、贪心法、回溯法等,理解算法的时间和空间复杂性以及复杂性分析等重要概念。典型的课程是"算法设计与分析"。

第三层次是"算法高级教学阶段",讲授工程应用中和数据智能处理相关的一些重要算法和模型,如最优化方法(如梯度下降法)、遗传算法、神经网络算法等。典型的课程是"工程最优化方法"、"模式识别"、"人工智能"等。

6.4 有特点的数

6.4.1 最小值和最大值

从序列中求出其中的最大值或最小值,通常采用"打擂台"算法。声明变量 Max 存储当前的最大值(擂主),依次与其余的数进行比较,当有比当前最大值还大的数,更新最大值,直到整个序列比较完毕,Max 的值为最大值。

【例 6-3】 随机产生 10 个 100～200 之间的数,求其最大值和最小值。

```
import random
x=random.choice(range(100,200))
max=100;min=200
for i in range(1 ,11):
    x=random.choice(range(100,200))
    print x,
    if x>max:
        max=x
    if x<min:
        min=x
print
print "max=", max
print "min=", min
```

运行结果如图 6.10 所示。

```
C:\WINDOWS\system32\cmd.exe - PAUSE
104 152 197 190 123 173 198 192 165 163
max= 198
min= 104
请按任意键继续. . .
```

图 6.10　例 6-3 程序运行结果

6.4.2　完全数

完全数,又称为完数,是具有以下特征的整数:该数所有的因子(除去其本身外)相加之和等于其自身。例如,整数 6 的因子为 1,2,3,6,除去整数 6 本身,其余的因子 1+2+3 之和与自身 6 相等,6 就是一个完数。

【例 6-4】　求出 1～1000 之间所有的完数。

```
for j in range(1,10001):
    s=0
    for i in range(1,j-1):
        if j %i==0:
            s=s+i
    if s==j:
        print j
```

程序运行结果如图 6.11 所示



图 6.11　“完全数”运行结果

6.4.3　水仙花数

水仙花数是一个三位数,其每个位上的数字的立方和等于该数字本身。例如,153＝1＊1＊1+5＊5＊5+3＊3＊3,故 153 是水仙花数。

【例 6-5】　求出水仙花数。

```
for x in range(100,1000):
    a=x//100
    b=(x-100*a)//10
    c=x-100*a-10*b
    if x==a * a * a+b * b * b+c * c * c:
```

```
        print x,
```

程序运行结果如图 6.12 所示。



图 6.12 “水仙花数”运行结果

6.4.4 与素数有关的数

1. 素数

素数又称为质数，是一个大于 2 且不能被 1 和其本身以外的整数整除的整数。

【解析】 若 N 是素数，只能被 1 和 N 自身整除，即不能被 2,3,…,N－1 整除。根据一个命题的逆否命题等于其本身的定律。只要 2,3,…,N－1 之中有一个数能被 N 整除，N 就不是素数；反之，如果 2,3,…,N－1 之中没有一个数能被 N 整除，N 就是素数。

【例 6-6】 判断从键盘上输入的整数是否为素数。

```
i=2
IsPrime=True
num=input("a number:")
for i in range(2,num-1):
    if num %i==0:
        IsPrime=False
        break                        #break的作用是什么?
if IsPrime==True:
    print num,"is prime"
else:
    print num,"is not prime"
```

假设从键盘输入了 9，程序运行过程如表 6.2 所示。

表 6.2 程序运行过程

变量 i	表达式 num % i	布尔值 IsPrime
2	1	true
3	0	false

如果没有 break 语句，程序将按表 6.3 运行。

表 6.3　没有 break 语句的程序运行过程

变量 i	表达式 num % i	布尔值 IsPrime
2	1	true
3	0	false
4	1	false
5	4	false
6	3	false
7	2	false
8	1	false

题意若变成求出 100 之内的所有素数，如何去做？

2. 孪生素数

孪生素数是指两个素数之差为 2，例如，3 和 5，5 和 7，11 和 13……

【例 6-7】 求 100 以内的孪生素数。

```
def isprime(num):              #用户自定义函数 isprime
    Prime=True
    for i in range(2,num-1):
        if num %i==0:
            Prime=False
            break
    if Prime==True:
        return num

for num in range(5,100):
    if isprime(num) and isprime(num+2):
        print num , "and " , num+2 , "is twin primes "
```

程序运行结果如图 6.13 所示。

```
C:\WINDOWS\system32\cmd.exe - PAUSE
5 and  7 is twin primes
11 and  13 is twin primes
17 and  19 is twin primes
29 and  31 is twin primes
41 and  43 is twin primes
59 and  61 is twin primes
71 and  73 is twin primes
请按任意键继续. . .
```

图 6.13　“孪生素数”运行结果

3. 回文素数

对一个整数 n(n≥11)从左向右还是从右向左读其结果值相同且是素数,即称 n 为回文素数。例如,11,101,131,151……

【例 6-8】 求 1000 以内的回文素数。

方法 1:先筛选出所有的素数,再在所找到的素数中找回文数。

```
def isPrime(value):
    isPrime=1
    for i in range(value-1, 2, -1):
        if value% i==0:
            isPrime=0
            break
    return isPrime

for i in range(2,1000):
    if isPrime(i)==1:
        if i<100:
            b=i//10
            c=i %10
            if b==c:
                print i,
        else:
            a=i // 100
            c=i%10
            if a==c:
                print i,
```

程序运行结果如图 6.14 所示。

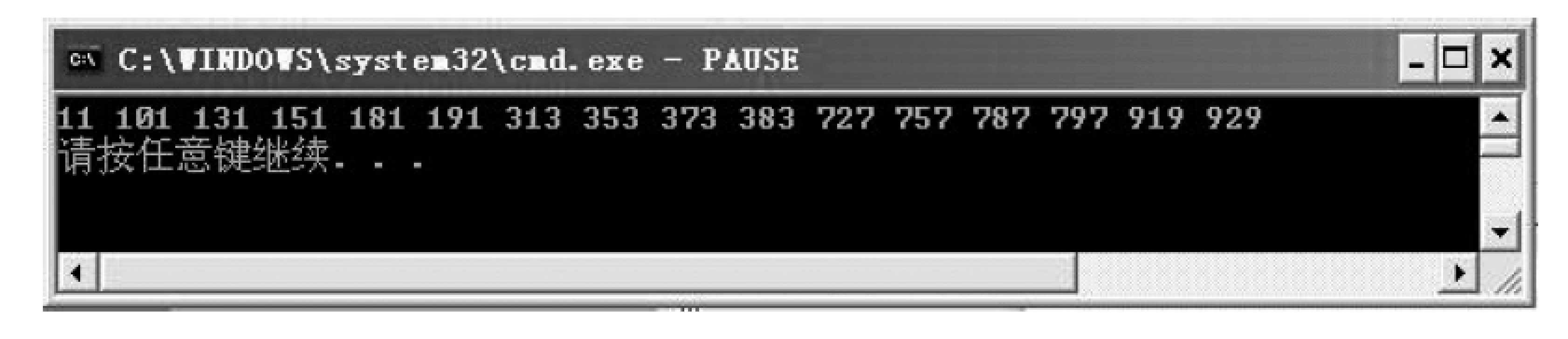


图 6.14 “回文素数”运行结果

方法 2:先找出回文数,再在回文数中找素数,此方案须遍历所有的回文数。读者可完成此代码,并分析两种方法的效率。

6.5 经典趣味题

6.5.1 鸡兔问题

【例 6-9】 鸡兔问题:鸡兔共有 30 只,脚共有 90 个,问鸡、兔各有多少只?

【解析】 设鸡为 x 只,兔为 y 只,根据题目要求,列出方程组为

$$\begin{cases} x + y = 30 \\ 2x + 4y = 90 \end{cases}$$

采用“试凑法”，将 x 和 y 的每一个取值代入方程组中进行尝试。

方法 1：利用二重循环来实现。

```
for x in range(0,31):
    for y in range(0,31):
        if (x+y==30 and 2 * x+4 * y==90):
            print "Chicken is ",x
            print "rabbit is ", y
```

注意：采用二重循环，循环体执行了 31×31=961 次。

方法 2：利用一重循环来实现。

```
for x in range(0,31):
    y=30-x
    if 2 * x+4 * y==90:
        print "Chicken is ",x
        print "rabbit is ", y
```

注意：采用一重循环，循环体执行了 31 次。

方法 3：假设鸡兔共有 a 只，脚共有 b 个，a 为 30，b 为 90。那么方程组为

$$\begin{cases} x + y = a \\ 2x + 4y = b \end{cases} \Rightarrow \begin{cases} x = (4a - b)/2 \\ y = (b - 2a)/2 \end{cases}$$

【代码】

```
a=30;b=90
x=(4 * a-b)//2
y=(b-2 * a)//2
print "Chicken is ",x
print "rabbit is ", y
```

6.5.2　百钱买百鸡

【例 6-10】　百元买百鸡问题。

公元 6 世纪末，我国古代数学家张丘建在《算经》里提出了一个不定方程问题，世界数学史上称为“百鸡问题”。内容如下：鸡翁一，值钱五，鸡母一，值钱三，鸡雏三，值钱一。百钱买百鸡，问鸡翁、母、雏各几何？翻译成现代文：公鸡每只 5 元，母鸡每只 3 元，小鸡三只 1 元。现在有 100 元钱要求买 100 只鸡，问小鸡、公鸡和母鸡各多少只？

【解析】　设公鸡、母鸡、小鸡各为 x, y, z 只，根据题目要求，列出方程为

$$\begin{cases} x + y + z = 100 \\ 5x + 3y + (1/3)z = 100 \end{cases}$$

三个未知数，两个方程，此题有若干个解。

采用三重循环来实现：

```
for x in range(0,101):
    for y in range(0,101):
        for z in range(0,101):
            if (x+y+z==100 and 5 * x+3 * y+1/3*z==100):
                print "roosters:" ,x,"hens:" , y,"chickens:" , z
```

程序运行结果如图 6.15 所示。

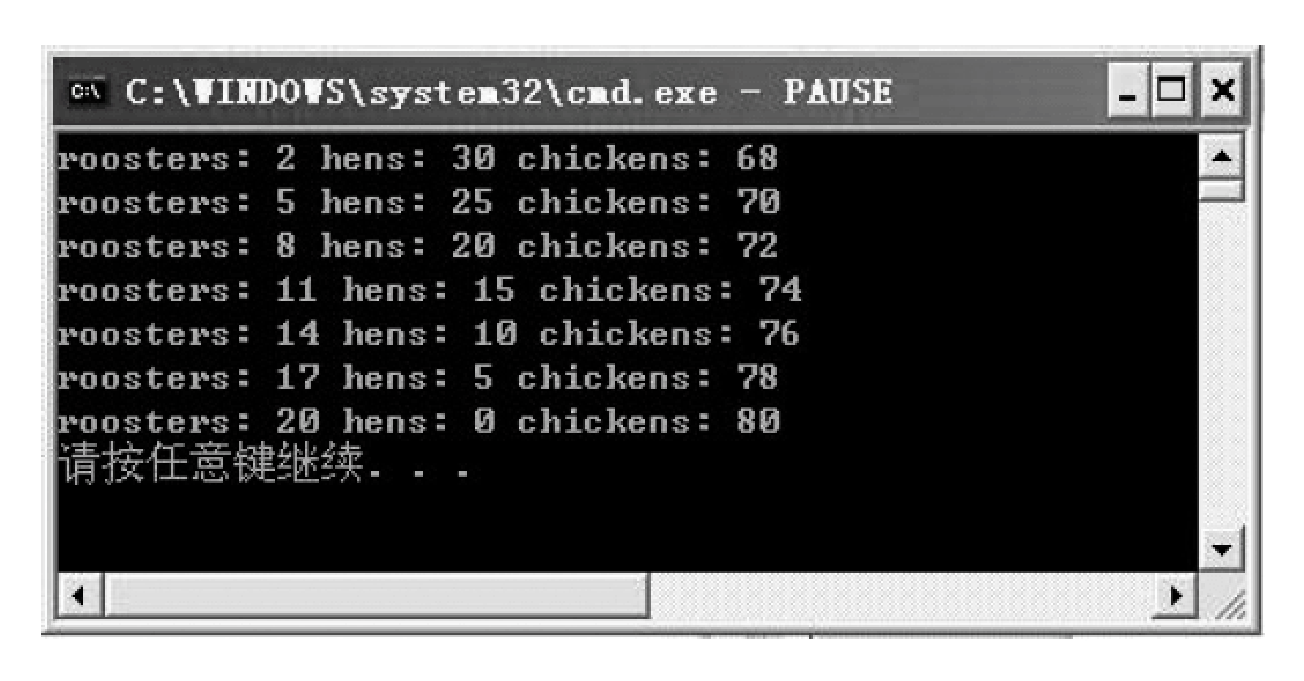


图 6.15 例 6-10 程序运行结果

仿照鸡兔问题,读者可以给出其他方法。

6.5.3 猴子吃桃

【例 6-11】 猴子吃桃问题。

猴子第一天摘下若干个桃子,当天吃了一半,还不过瘾,又多吃了一个。第二天早上将剩下的桃子吃掉一半,又多吃了一个。以后每天早上都吃了前一天剩下的一半还多一个。到第 10 天早上想再吃时,见只剩下一个桃子了。求第一天共摘多少桃子。

【解析】 这是一个“递推”问题,从最后一天推出倒数第二天的桃子,再从倒数第二天的桃子推出倒数第三天的桃子……

设第 n 天的桃子为 x_n,那么前一天 x_{n-1} 的桃子数为 $x_n=\frac{1}{2}x_{n-1}-1$,也就是:$x_{n-1}=(x_n+1)\times 2$。

```
total=1
days=10
for i in range(1 ,days):
    total= (total+1) * 2
print total
```

6.6 习题

1. 任意给一个四位数(各位数不完全相同),各个位上的数可组成一个最大数和一个最小数,它们的差又能组成一个最大数和一个最小数,直到某一步得到的差将会出现循环重复。写一个程序统计所有满足以上条件的四位数。

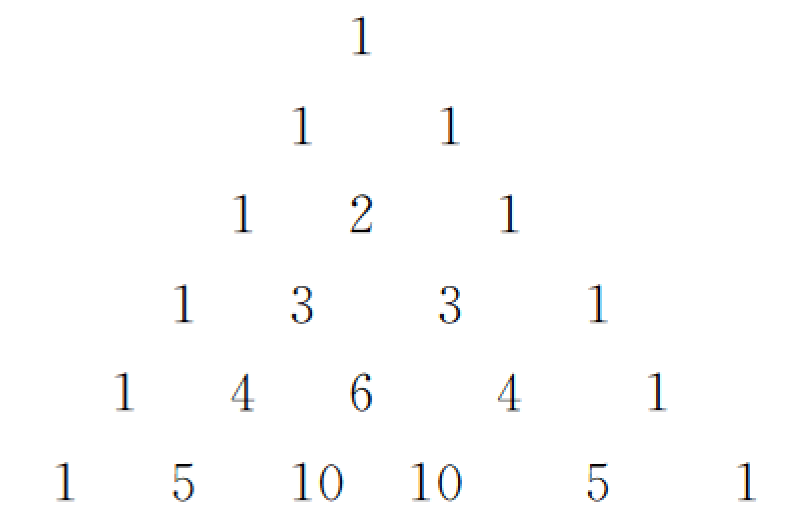
例如：3100－0013＝3087

8730－0378＝8352

8532－2358＝6174

7641－1467＝6174

2. $(a+b)$的 n 次幂的展开式中各项的系数很有规律，对于 $n=2,3,4$ 时，系数分别是：1 2 1，1 3 3 1，1 4 6 4 1。这些系数构成了著名的杨辉三角形：



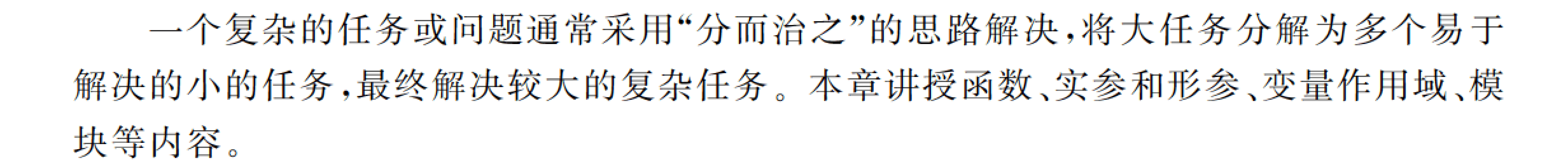
编写代码实现杨辉三角形。

3. 如果有两个数，每一个数的所有约数(除它本身以外)的和正好等于对方，则称这两个数为互满数。求出 3000 以内所有的互满数，并显示输出。

4. 验证任意一个大于 5 的奇数可表示为 3 个素数之和。

第7章

函数与模块



一个复杂的任务或问题通常采用“分而治之”的思路解决，将大任务分解为多个易于解决的小的任务，最终解决较大的复杂任务。本章讲授函数、实参和形参、变量作用域、模块等内容。

7.1 函数

7.1.1 函数概念

复杂的程序往往由多个较小的程序片段组成，将这些程序片段所执行的相同功能分离出来作为公共的独立单位使用，只须在程序的不同地方调用执行即可，这个独立公共单位称为函数。

Python 的函数分为系统函数和用户自定义函数。系统函数又称内置函数或内建函数。用户自己创建函数称为用户自定义函数。一般来说，函数的大小应在 70～200 行代码之间，如果小于这个范围，就要考虑这个函数是否需要单独提出来，如果大于这个范围，就应当考虑是否应将大的函数细化。

函数具有以下好处：

(1) 简化程序的结构，提高程序的可读性。

(2) 函数代码可以重复调用，减少了程序中大量重复代码的书写，函数一旦创建，便可以在程序的任何一个地方调用。

(3) 使得应用程序更容易调试、修改和维护。

(4) 便于多人协同合作开发。

7.1.2 函数声明和调用

在 Python 中，函数声明语法格式为

```
def<函数名>([<形参列表>]):
    [<函数体>]
```

说明：

(1) 函数使用关键字 def(define 的缩写)声明，函数名为有效的标识符，形参列表为

函数的参数。

(2) 函数没有明显的 begin 和 end，没有标明函数的开始和结束的花括号。唯一的分隔符是一个冒号(:)。

(3) 函数名下的每条语句前都要用 Tab 键缩进，没有缩进的第一行则被视为在函数体之外的语句，与函数同级的程序语句。

(4) 函数没有定义返回的数据类型，函数通过 return 语句返回指定的值，否则将返回空值(None)。

【例 7-1】 函数声明。

```
def sayHello():
    print 'hello world!'
```

【解析】 sayHello 是函数的名称，后面的括号里是参数，这里没有，表示不需要参数。但括号和后面的冒号都不能少。

缩进的代码块(print 'hello world! ')是函数的内容，称作函数体。

【例 7-2】 利用海伦公式，求三角形面积。

```
#triarea.py
import math
def triarea(x,y,z):
    s= (x+ y+ z) / 2
    print math.sqrt((s- x)  *  (s- y)  *  (s- z)  *  s)
#主函数
triarea(3,4,5)
```

【解析】 triarea(3,4,5)调用 triarea(x,y,z)，程序执行步骤如图 7.1 所示。

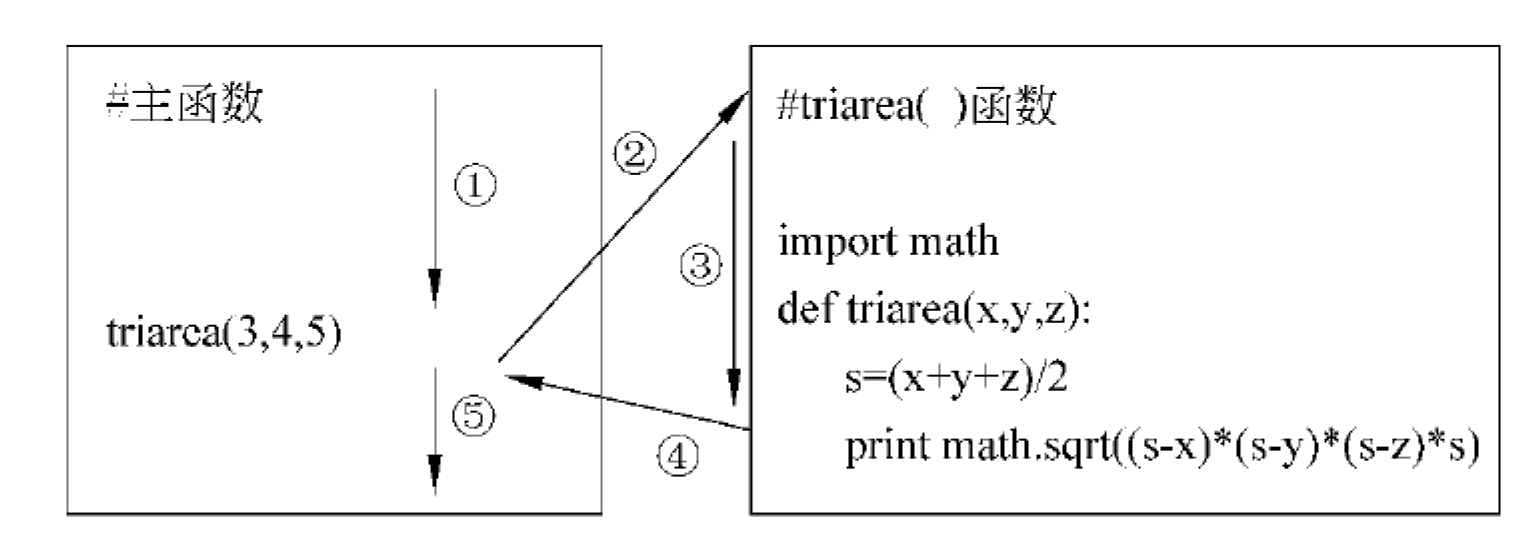


图 7.1　函数调用

函数调用步骤如下：

步骤 1： 运行到 triarea(3,4,5)语句时，主函数中断，Python 寻找同名的 triarea()函数。如果没有找到，Python 提示语法错误。

步骤 2：找到同名函数，进行函数调用，实现参数的传递，如图 7.1②箭头。

triarea(3,4,5)中 3,4,5 是实参的取值。

triarea(x,y,z)中 x,y,z 是形参。

在实参和形参的结合时，必须遵循以下三条规则：

(1) 实参和形参个数相等。

(2) 实参和形参类型依次相等。

(3) 实参给形参依次传递，实参和形参传递如表 7.1 所示。

表 7.1 函数调用时，实参和形参传递的三条规则

三条规则	实参(3,4,5)	形参(x,y,z)	运行结果
参数个数	3 个	3 个	个数相等
参数类型	3 为整型 4 为整型 5 为整型	x 为整型 y 为整型 z 为整型	依次类型相同
依次传递			则 x 得到 3，y 得到 4，z 得到 5

步骤 3：执行海伦公式函数，如图 7.1 的③箭头。

步骤 4：海伦公式执行结束，程序返回到主函数的中断处，如图 7.1 的④箭头。

步骤 5：继续执行主函数，如图 7.1 的⑤箭头。

7.1.3 实参和形参

实参(实际参数)是指传递给函数的值，即在调用函数时，由调用语句传给函数的常量、变量或表达式。

形参(形式参数)是在定义函数时，函数名后面括号中的变量，用逗号分隔。作为函数与主调程序交互的接口，用来接收调用该函数时传递的实参，从主调程序获得初值，或将计算结果返回给主调程序。

形参和实参具有以下特点：

(1) 函数在被调用前，形参只是代表了执行该函数所需要参数的个数、类型，并没有具体的数值，形参只能是变量，不能是常量、表达式。只有当调用时，主调函数将实参的值传递给形参，形参才具有值。

(2) 形参只有在被调用时才分配内存单元，调用结束后释放内存单元，因此形参只在函数内部有效，函数调用结束返回主调用函数后则不能再使用该形参变量。

(3) 实参可以是常量、变量、表达式、函数等，无论实参是何种数据类型的变量，函数调用时必须是确定的值，以便把这些值传送给形参。

(4) 实参和形参在数量、类型、顺序方面应严格一致，否则会发生类型不匹配错误。

7.1.4 引用传参

Python 语言的参数传递是引用传递，即被调用函数中修改了形式参数值，调用函数的实际参数值也被改变。函数调用时，调用函数把实参变量的“地址”传给形参，整个执行期间实参和形参共用同一地址的存储单元，实参和形参其实就是一个，被调函数对形参的任何操作都等同于对实参的操作，因此实参值会随着被调用函数形参值的改变而改变，参数值的传递是“双向”，如图 7.2 所示。

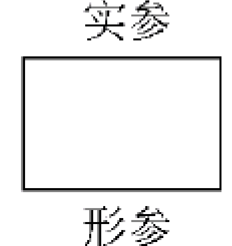


图 7.2 引用传参

【例 7-3】　引用举例。

```
def SwapRef(x,y):
    t=x
    x=y
    y=t
主函数
a=10
b=20
print "exchange before a=",a ,"b=", b
SwapRef(a, b)
print "exchange after a=",a ,"b=", b
```

输出结果如图 7.3 所示。

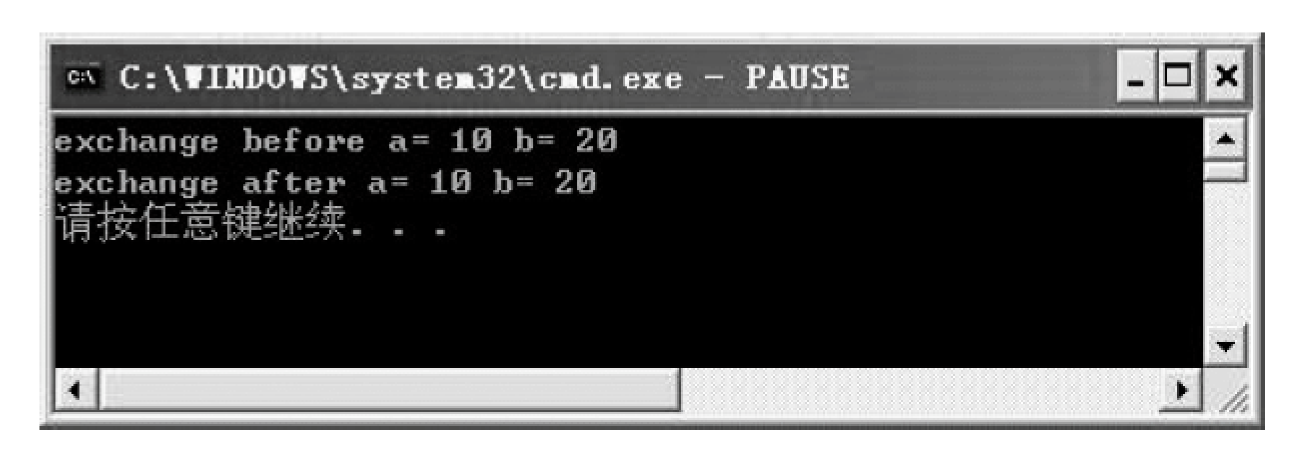


图 7.3　例 7-3 程序运行结果

7.1.5　return 语句

return 语句用来从一个函数返回，即跳出函数，同时从函数返回一个值。

【例 7-4】　return 举例。

```
def maximum(x, y):
    if x>y:
        return x
    else:
        return y
#主函数
print maximum(2, 3)
```

【解析】　maximum 函数用于判断两个数中的较大者，并返回参数中的最大值。

7.1.6　函数是对象

Python 语言中的一切都是对象，因此函数也是一个对象，可以像普通对象一样使用，把一个函数赋值给另一个变量。

【例 7-5】　对象举例。

```
def printMax(a, b):
    if a>b:
        print a, "is maximum"
```

```
    else:
        print b, "is maximum"
print type(printMax)
anotherPrint=printMax
printMax(1, 10)
anotherPrint(1, 10)
print type(anotherPrint)
```

输出结果如图 7.4 所示。



图 7.4 例 7-5 程序运行结果

7.2 参数类型

Python 的参数分为必备参数、默认参数、关键参数和可变长参数等。

7.2.1 必备参数

必备参数须以正确的顺序传入函数，调用时参数的数量必须和声明时一样。

【例 7-6】 必备参数举例。

```
def printme(str):
    print str;
    return;
#主函数
printme();
```

输出结果：

```
Traceback (most recent call last):
    File "test.py", line 11, in<module>
        printme();
TypeError: printme() takes exactly 1 argument (0 given)
```

【解析】 printme()函数必须传入一个参数，不然会出现语法错误。

7.2.2 默认参数

默认参数是指允许函数参数有默认值，如果调用函数时不给参数传值，参数将获得默

认值。Python 通过在函数定义的形参名后加上赋值运算符(＝)和默认值,给形参指定默认参数值。

注意：默认参数值是一个不可变的参数。

【例 7-7】　使用默认参数值。

```
def say(message, times=1):
    print message * times
主函数
say('Hello')        # 默认参数 times 为 1
say('World', 5)
```

输出

```
Hello
WorldWorldWorldWorldWorld
```

7.2.3　关键参数

函数的多个参数值一般默认从左到右依次传入。但是,Python 也提供了灵活的传参顺序,引入了关键参数用于改变赋值顺序,关键参数又称命名参数,可以以任意顺序指定参数。

【例 7-8】　使用关键参数。

```
def func(a, b=5, c=10):
    print "a is", a, "and b is", b, "and c is", c
主函数
func(3, 7)
func(25, c=24)
func(c= 50, a=100)
```

输出

```
a is 3 and b is 7 and c is 10
a is 25 and b is 5 and c is 24
a is 100 and b is 5 and c is 50
```

7.2.4　可变长参数

可变长参数又称不定长参数,若参数以一个 * 号开头的代表一个任意长度的元组,可以接收连续一串参数。参数以两个 * 号开头的代表一个字典,参数的形式是“key＝value”,接受连续任意多个参数。

【例 7-9】　可变长参数举例。

```
def foo(x, * y, * * z):
    print x
```

```
    print y
    print z
```

主函数分别有如下三种形式：

```
foo(1)
```

输出结果如图 7.5 所示。

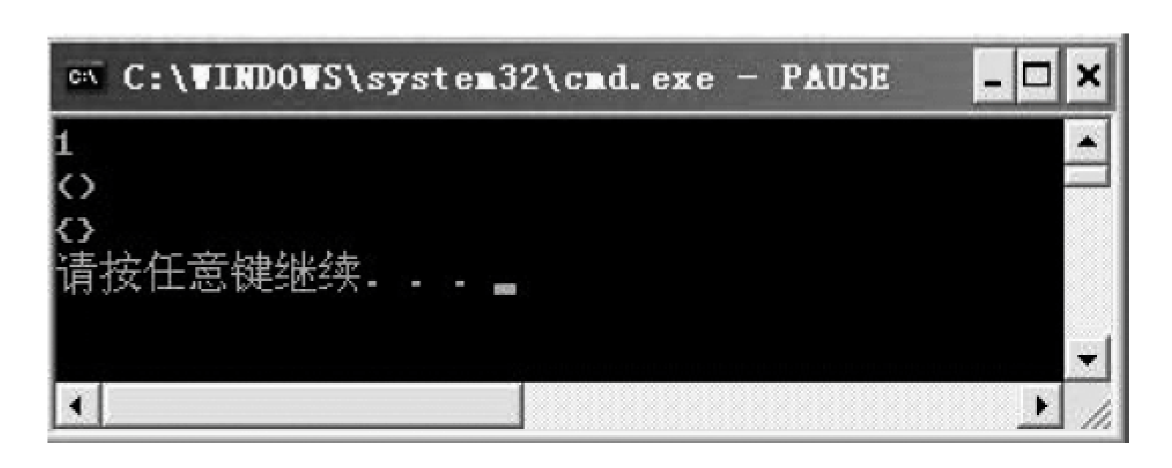


图 7.5　例 7-9 程序运行结果 1

```
foo(1,2,3,4)
```

输出结果如图 7.6 所示。

```
C:\WINDOWS\system32\cmd.exe - PAUSE
1
(2, 3, 4)
{}
请按任意键继续. . .
```

图 7.6　例 7-9 程序运行结果 2

```
foo(1,2,3,a="a",b="b")
```

输出结果如图 7.7 所示。



图 7.7　例 7-9 程序运行结果 3

7.3　两类特殊函数

7.3.1　lambda 函数

lambda 函数是一种简单的匿名函数，用于函数速写的作用，允许在使用的代码内嵌入一个函数的定义。lambda 是可选的，能够用 def 替代。

lambda 函数的形式如下：

```
labmda 参数：表达式
```

lambda 函数默认返回表达式的值。

【例 7-10】 lambda 函数举例。

```
>>> f=lambda a,b: a+b
>>> f(1,2)
3
>>> f("abc", "def")
"abcdef"
```

【解析】 f 等价于 def f(a,b)：return a+b。

7.3.2 递归函数

【例 7-11】 计算 4 的阶乘。

方法一：4 的阶乘为 4×3×2×1。

```
s=1
for i in range(1,5):
s=s * i
print s
```

方法二：4 的阶乘为 4 乘以 3 的阶乘，3 的阶乘为 3 乘以 2 的阶乘，依此类推。

```
def fact(n):
    if n==1:
        return 1
    return n * fact(n-1)
主函数
    Print fact(4)
```

【解析】 方法一通过循环语句来计算阶乘，该方法的前提是了解阶乘的计算过程，并可用语句把计算过程模拟出来。方法二不直接找到计算 4 的阶乘的方法，而是试图找到 4 的阶乘和 3 的阶乘之间的递推关系，通过递推关系将原来问题缩小成一个规模更小的同类问题，并延续这一缩小规模的过程，直到在某一规模上（当 n 为 1 时）问题的解是已知。方法二解决问题的思想称为递归。

fac(n)递归求解如图 7.8 所示。

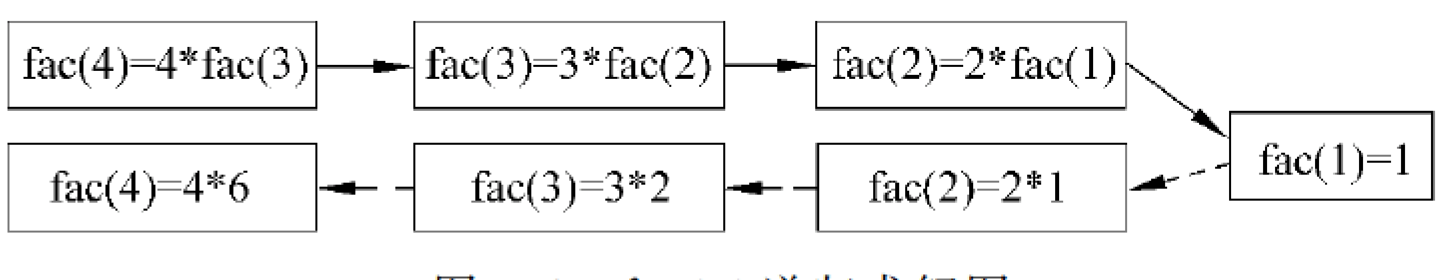


图 7.8 fac(n)递归求解图

递归函数就是在自定义函数内部调用其自己，包含了递推和回归两个过程。构成递归的条件是：

(1) 递归结束条件和结束时的值。

(2) 能用递归形式表示，并且递归向结束条件发展。

【例 7-12】 利用递归求最大公约数。

最大公约数可以采用“辗转相除法”实现，递归公式如下：

$$\gcd(m,n)=\begin{cases} n & m \bmod n = 0 \\ \gcd(n, m \bmod n) & m \bmod n \neq 0 \end{cases}$$

```
def gcd(m,n):
    if m %n==0:
        return n
    else:
        return gcd(n, m%n)
主函数
M=input("m=")
N=input("n=")
print gcd(M, N)
```

输出结果如图 7.9 所示。



图 7.9 例 7-12 程序运行结果

7.4 变量作用域

变量作用域是指变量有效可用的范围，Python 与大多数程序语言一样有局部变量和全局变量，变量的声明通过首次赋值产生，类似于 Visual Basic 变量的隐式声明，当超出变量作用范围时会自动消亡。

7.4.1 局部变量

局部变量是指定义在函数体内的变量，只能被本函数使用，与函数外具有相同名称的其他变量没有任何关系。

【例 7-13】 局部变量举例。

```
def func(x):
    print "x is", x
    x=2
    print "changed local x to", x
```

```
#主程序
x=50           #局部变量
func(x)
print "x is still", x
```

输出结果如图 7.10 所示。

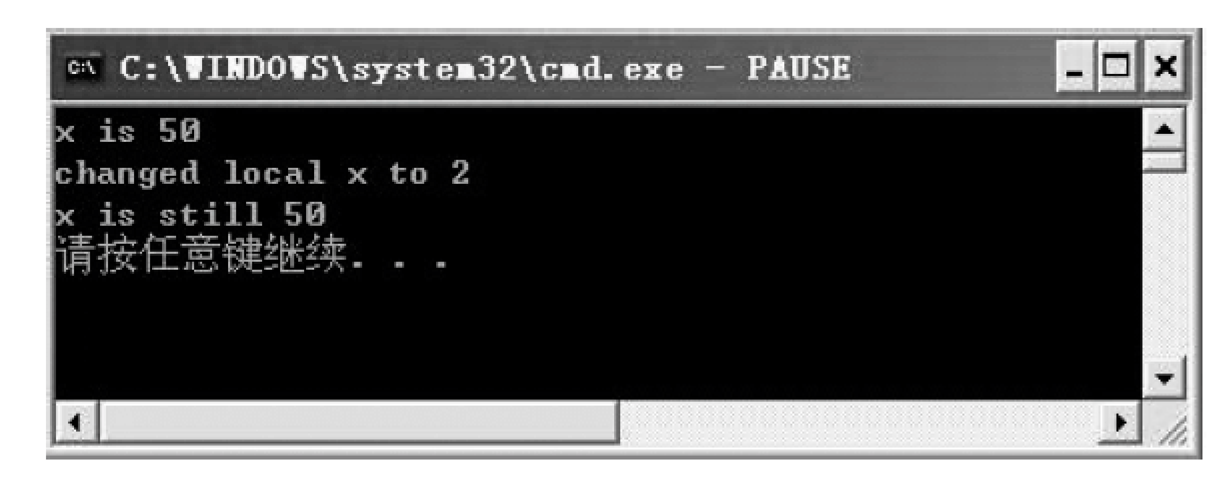


图 7.10　例 7-13 程序运行结果

【解析】

步骤 1：主函数中，给 x 赋值为 50。

步骤 2：Func 函数里，x 是函数的局部变量，给 x 赋值为 2。

步骤 3：返回主函数，print 语句 x 的值没有受到 func 函数里对 x 值的改变，说明主函数中 x 不受影响。

7.4.2　全局变量

全局变量是指定义在函数体外的变量，也称为公用变量，可在其他模块和函数中使用，全局变量声明使用关键字 global。可以使用 global 语句指定多个全局变量，如 global x，y，z。

【例 7-14】　全局变量举例。

```
def func():
    global x
    print "x is", x
    x=2
    print "Changed global x to", x
#主函数
x=50
func()
print "Value of x is", x
```

输出结果如图 7.11 所示。

```
C:\WINDOWS\system32\cmd.exe - PAUSE
x is 50
Changed global x to 2
Value of x is 2
请按任意键继续. . .
```

图 7.11　例 7-14 程序运行结果

【解析】 global 语句被用来声明 x 是全局变量，在 func 函数内改变 x 的值为 2 时，这个变化也反映在主函数中 x 的值。

7.5 模块

7.5.1 命名空间

Python 中的所有代码都与一个命名空间关联。所谓的命名空间可以理解为一个容器，容器内装载许多标识符，不同容器中同名的标识符不会相互冲突。

Python 的命名空间具有三条规则：

(1) 赋值产生标识符，赋值的地点决定标识符所处的命名空间。

(2) 函数定义产生新的命名空间。

(3) Python 搜索一个标识符按照四层命名空间的顺序，即“LEGB”。

① L(local)，表示在一个函数定义中，而且在这个函数里面没有再包含函数的定义。

② E(enclosing function)，表示在一个函数定义中，但这个函数里面还包含函数的定义，L 层和 E 层相对而言。

③ G(global)，是指一个模块的命名空间，也就是说在一个. py 文件中定义的标识符，但不在一个函数中。

④ B(builtin)，Python 解释器启动时会自动载入__builtin__模块，这个模块有 list 和 str 等内置函数。

7.5.2 模块定义与导入

模块是最高级别的程序组织单元，将程序代码和数据封装起来以便重用。模块比函数粒度更大，一个模块可以包含若干个函数。与函数相似，模块也分系统模块和用户自定义模块，用户自定义的一个模块就是一个 py 文件。

模块的导入有如下方法：

方法 1：引入模块

```
import moduleName
```

方法 2：引入模块下的函数

```
from moduleName import function1, function2
```

方法 3：引入模块的所有函数

```
from moduleName import *
```

【例 7-15】 采用模块导入的方法 1。

```
#numbers.py
def divide(a, b):
    q=a/b
```

```
    r=a-q*b
    return q, r     #q 为商,r 为余数

#最大公约数
def gcd(x, y):
    g=y
    while x>0:
        g=x
        x=y % x
        y=g
    return g

#主函数
import numbers
x, y=numbers.divide(11,7)
print x,y
n=numbers.gcd(8, 7)
print n
```

输出结果如图 7.12 所示。



图 7.12　例 7-15 程序运行结果

注意：numbers.py 模块必须与 main.py 放置在同一个目录下。

【解析】　numbers 模块的导入使用了方法 1(import numbers)，import 语句创建一个新的名字空间，该空间包含模块中所有的对象，当访问这个名字空间时，需将模块名作为前缀使用，如 numbers.divide()，numbers.gcd()。

#使用一个不同的模块名字访问这个模块。

```
import numbers as num
x, y=num.divide(11,7)
print x,y
n=num.gcd(8, 7)
print n
```

【例 7-16】　采用模块导入的方法 2。

```
from numbers import divide,gcd
x, y=divide(11,7)
print x,y
```

```
n=gcd(8, 7)
print n
```

【解析】 采用方法 2 导入指定的对象到当前的名称空间，使用 from 语句，模块名不用作为前缀。

7.6 习题

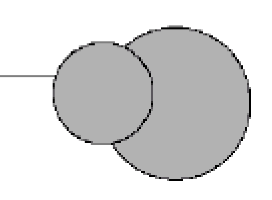
1. 编制判断素数的 Sub 函数或 Function 函数，验证哥德巴赫猜想：一个不小于 7 的偶数可以表示为两个素数之和。例如，7＝3＋3，8＝3＋5，10＝3＋7。

2. 实现求两数中较大数的函数。

3. 实现计算表达式 1＋3＋…＋(2n－1)值的函数。

4. 完成一函数，将所给的(1,2,3,－5,－4,5,9,－8,－1)重新排列，使得所有负数都在正数的左边。

第 8 章



面向对象程序设计基础

本章首先讲述了面向对象编程(OO,Object-Oriented)的基本概念,如对象和类、对象的三大特性。其次,就 Python 语言如何实现 OO 的类和对象,类属性和实例属性、构造函数和析构函数、继承性、多态性等进行了讲解。

8.1　面向对象概述

8.1.1　基本概念

Python 语言涉及模块、语句、表达式、对象等概念。它们的关系如下所示：程序由模块构成;模块由语句构成;语句由表达式构成;表达式由对象构成。Python 是完全面向对象的程序设计语言,认为一切东西都是对象,如一个人、一张桌子、一支笔等,Windows 和 ubuntu 操作系统环境中的窗口、命令按钮、文本框等也都是对象。

现实世界中的实体映射为计算机世界中的对象,现实世界中的实体抽象为概念世界中的抽象数据类,概念世界中的抽象数据类映射为计算机世界中的类。在计算机世界中,类是对象的抽象,而对象是类的实例化。现实世界、概念世界与计算机世界中类与对象的映射关系如图 8.1 所示。

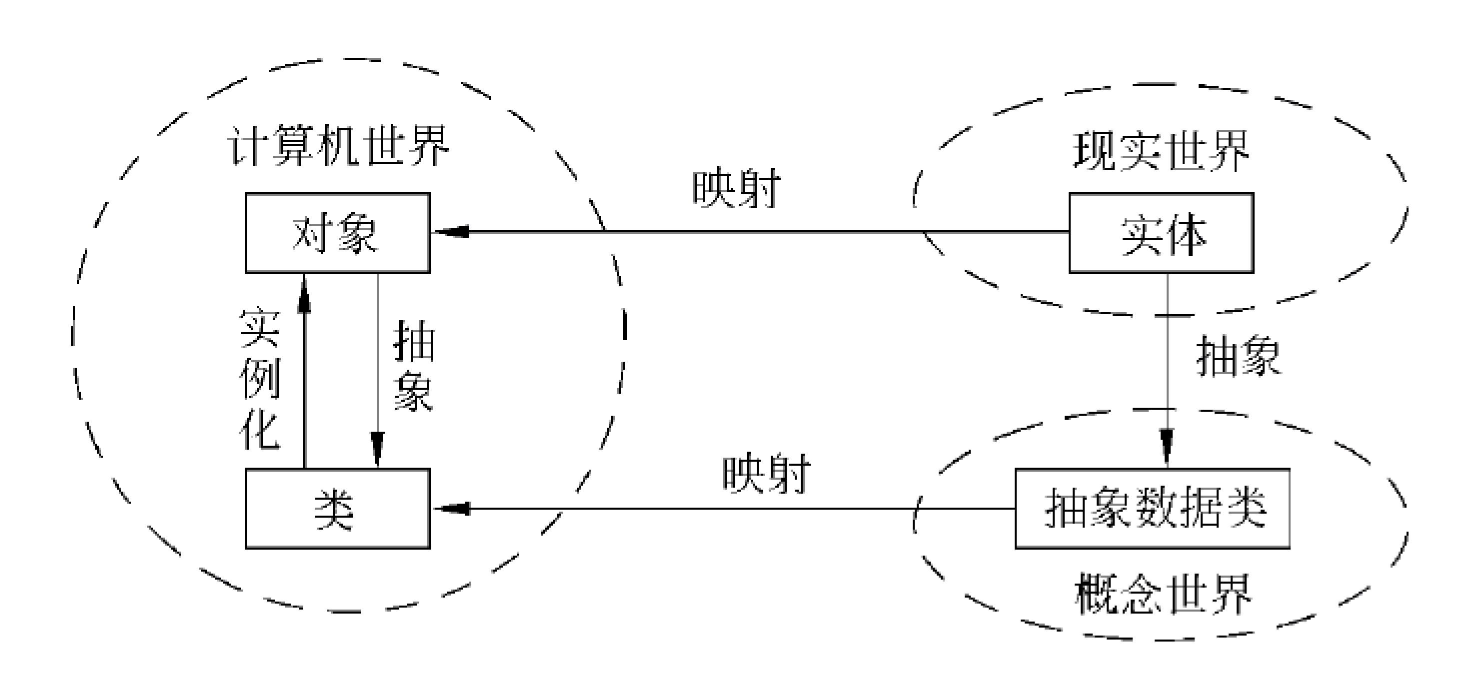


图 8.1　三个世界的映射关系

不同类的对象具有不同的属性、方法及其能够响应的事件。以人为例分析类和对象。人类是所有人的抽象,它不是一个具体的人,只是概念上的人,例如北京猿人、陕西人等。将人类实例化为对象,如张三、男、1984 年 9 月出生、身高 180cm、体重 78 公斤,张三就是一个具体的对象,具有姓名、性别、出生日期、身高、体重等属性;张三具有思考、学习 Python 等行为;张三对于奖励或惩罚等外部事件会做出不同的反应,如高兴、悲伤等。

8.1.2 与面向过程不同

面向过程解决问题的方法就是按照事件发展过程进行,把解决问题的步骤列出,然后一步一步执行。而面向对象思想解决问题的方式是模拟人认识世界的思维方法,把与问题相关的数据提取出来,将具有相同属性的事物抽象为"类",设计出"类"的方法,程序执行时,将"类"实例化为"对象",调用"类"的方法解决问题。

【例 8-1】 采用 OP 和 OO 设计用户登录程序。

面向过程(Procedure-Oriented)程序设计思路如下:

(1) 创建一个让用户输入用户名和密码的应用程序界面。

(2) 用户提交数据后首先判断用户名和密码是否为空,若为空,提示出错,否则继续。

(3) 判断输入的用户名是否为合法用户名。若合法,则继续,否则提示用户名错。

(4) 判断输入的密码是否正确。若正确,则登录成功,否则提示密码错。

面向对象(Object-Oriented)程序设计思路如下:

(1) 创建用于输入用户名和密码的应用程序界面。

(2) 将用户看成一个对象。

(3) 用户对象拥有一个用于检查用户名和密码合法性的方法。

(4) 用户提交数据后,调用方法对数据进行检验,并根据检验返回结果确定用户登录是否成功。

面向过程和面向对象方法都能完成用户登录程序。面向过程要求用户必须按照程序规定好的步骤一步一步操作,不能任意跳跃。面向对象是事件驱动,程序运行由消息的产生和处理展开,而消息的产生不会以任何预定义的次序出现,往往没有先后次序。

因此,面向对象的程序设计方法具有以下一些优点:

(1) 可扩展性。在面向过程的设计方法中功能的实现分散在了许多步骤中,这对功能的扩展极为不利。而在面向对象的设计中,功能靠方法来实现,需要新功能时只需要创建新的方法即可。

(2) 分工明确。面向对象的设计方法中将所有问题划分成对象,程序功能依靠方法来实现,具有明确的分工。当因对象发生变化需要修改程序时,通过局部改动就可以完成,保证了程序具有较好的可移植性。

8.1.3 面向对象三大特性

面向对象有封装性、继承性和多态性三大特性,下面依次进行介绍。

1. 封装性

类是对客观事物的抽象,是具有一组相同的属性和操作的对象的集合。例如,对于不同的教师,无论是王燕还是李梅都拥有姓名、性别、年龄、职称和教龄等属性(数据),都拥有讲课、批改作业等行为能力,因此,可将王燕、李梅等具体教师对象抽象为教师类。封装具有对内部细节隐藏保护的能力,防止外部程序破坏类的内部数据,便于程序的维护和修改。

封装性使得面向对象具有抽象性。抽象性是指将具有一致的数据结构(属性)和行为(操作)的对象抽象成类,用于反映与应用有关的重要性质,而忽略其他一些无关内容。

2. 继承性

继承是一种连接类与类的层次模型,利用现有类派生出新类的过程称为继承。新类拥有现有类的特性,又增加了自身新的特性。例如,动物物种系统中,基类是“动物”,可以派生出许多更加特殊的动物类,如脊椎动物,爬行动物、哺乳动物等,它们在拥有基类所有属性和操作的基础上还有其各自的属性和操作,如脊椎动物具有脊椎、爬行动物能够爬行、哺乳动物能够哺乳养育下一代等。继承可以简化类和对象的创建工作量,增强代码的可重用性。

对于一个派生类,如果只有一个基类,称为单继承。如果同时有多个基类,称为多重继承。单继承可以看成多重继承的简单特例,而多重继承可以看成多个单继承的组合。图 8.2(a)是单继承,运输汽车类和专用汽车类就是从汽车类派生而来;图 8.2(b)则是多继承,孩子类从母亲类和父亲类两个类派生而来。

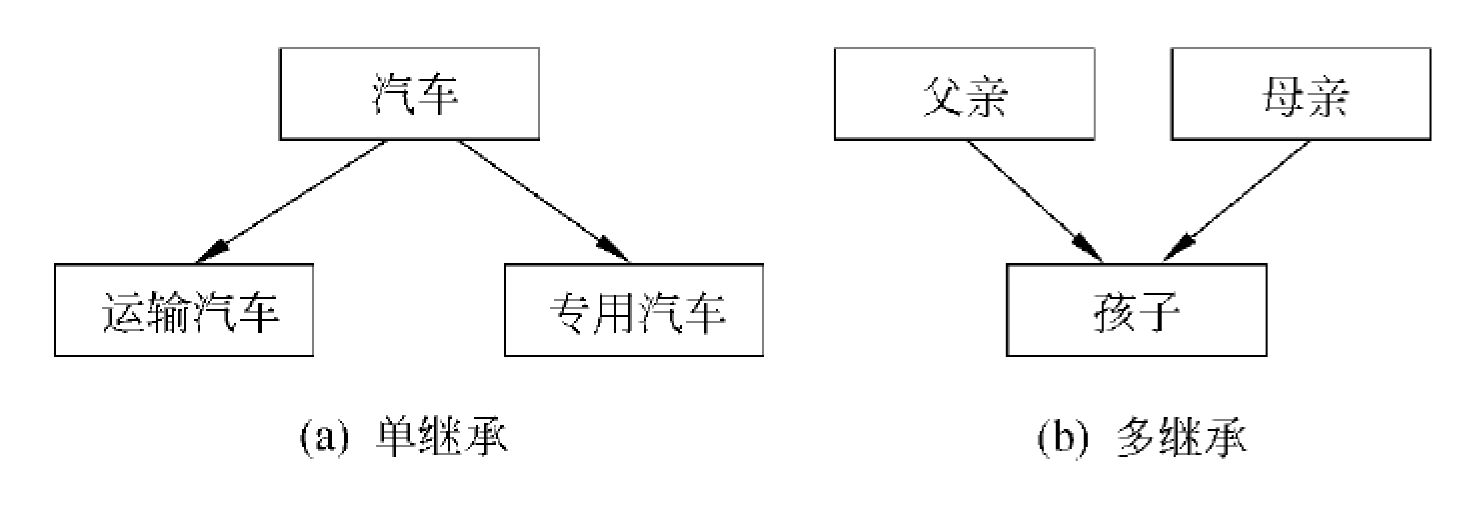


图 8.2 单继承与多继承

3. 多态性

多态性(polymorphism)一词来源于希腊语,“poly”表示多的意思,“morphos”表示形态的意思,“polymorphism”是指同一种事物具有多种形态。在自然语言中,多态性是“一词多义”,是指相同的动词作用到不同类型的对象上。例如,驾驶摩托车、驾驶汽车、驾驶飞机、驾驶轮船、驾驶火车等都具有相同的动作——“驾驶”,但其驾驶的对象和驾驶动作不同。

多态性是指两个或多个对象对于同一消息作出不同响应的方式。例如,某个属于“形体”基类的对象,在调用它的“计算面积”方法时,程序会自动判断出“形体”类型,如果是圆,则调用圆的“计算面积”方法,如果是正方形,则调用正方形的“计算面积”方法。多态性允许每个对象以适合自身的方式去响应共同的消息,为软件开发和维护提供了极大的方便。

8.2 类和对象

Python 使用 class 关键字构造类,并在类中定义属性和方法。通常认为类是对象的模板,对象是类创建出来的产品,对象是类的实例。

声明类的语法格式如下：

```
class 类名:
属性定义    #变量定义
方法定义    #函数定义
```

说明如下所示：

(1) 定义类的关键字为“Class...”,类名通常是第一个字母大写。

(2) 对象的定义通过使用类名后跟一对圆括号来创建。

(3) 类具有属性和方法。

【例 8-2】 Python 中的类和对象。

```
class Person:                    #声明 class 类
    name="zhou"                  #类属性
    age=80                       #公有类属性
    def sayHi(self):             #sayHi 方法
        print 'Hello, how are you? '

Print person.name
print Person.age                 #类属性,直接通过类名引用
p=Person()                       #创建对象 p
p.sayHi()
输出
    70
    Hello, how are you?
```

8.3 类属性与实例属性

Python 的属性有两种,一种是实例属性,另一种是类属性。

8.3.1 实例属性

实例属性作为实例对象的属性,只为单独的特定的对象所拥有。一般有两种方式定义：方式一,实例属性不在类中,而在类外显示定义;方式二,实例属性在类中的构造函数 _ _init_ _中定义,定义时以 self 作为前缀。

【例 8-3】 实例属性举例。

```
class People:
    name='jack'                   #类属性 name

p=People()
p.age=12                          #实例属性 age 在类 People 之外定义
print p.name                      #正确
print p.age                       #正确
```

```
print People.name              #正确
print People.age               #错误
```

【解析】 类 People 只有类属性 name，声明实例对象 p，age 作为对象 p 的实例属性，没有在类中显示定义。实例属性 age 是实例对象 p 所特有，类 People 并不拥有它，不能通过类来访问。

【例 8-4】 实例属性举例。

```
class People:
    name='jack'
    #__init__()是内置的构造方法，在实例化对象时自动调用
    def __init__(self,age):
        self.age=age

p=People(12)
print p.name                   #正确
print p.age                    #正确
print People.name              #正确
print People.age               #错误
```

【解析】 类 People 只有类属性 name，声明实例对象 p，age 作为对象 p 的实例属性，在构造函数__init__中定义，定义时以 self 作为前缀。

8.3.2　类属性

类属性是在类中方法之外定义的属性，又分为公有属性和私有属性，不像 C++ 通过 public 和 private 关键字区别公有属性和私有属性，Python 是以属性命名方式来区分，如果在属性名前面加了两个下划线“__”，则表明该属性是私有属性，否则为公有属性。

(1) 访问类属性

① 公有属性为类的所有对象共有，在类外可以通过类名和实例对象名两种方式访问。

- 通过对象名访问类属性，实例属性会强制屏蔽掉类属性，给出实例属性的数值。
- 通过类名访问类属性，将给出类属性的数值。

② 私有属性不能在类外通过类名和对象名访问，使得程序代码健壮。

【例 8-5】 类属性举例。

```
class People:
    name='jack'             #公有的类属性
    __age=12                #私有的类属性

p=People()
print p.name                #正确
print People.name           #正确
print p.__age               #错误，不能在类外通过实例对象访问私有的类属性
```

```
print People.__age           #错误,不能在类外通过类访问私有的类属性
```

(2) 修改与删除类属性

类属性修改必须通过实例对象,类属性的修改会产生一个同名的实例属性副本,类属性的修改实际是实例属性副本的修改,而不是类属性本身,不会影响到类属性数值,从而保护了类属性。

当实例属性被删除后,通过实力属性所访问的数值就是类属性的数值。

【例 8-6】 修改类属性。

```
class People:
    country='china'          #类属性

print People.country         #输出类属性的数值为'china'
p=People()                   #类的实例——对象 p
print p.country              #输出类属性的数值为'china'
p.country='japan'            #修改实例属性的数值为'japan'
print p.country              #实例属性会屏蔽同名的类属性,输出的数值为'japan'
print People.country         #输出类属性的数值为'china'
del p.country                #删除实例属性的数值 'japan'
print p.country              #输出类属性的数值为'china'
```

程序运行结果如图 8.3 所示。



图 8.3 例 8-6 程序运行结果

8.4 方法

Python 方法分为对象方法、类方法和静态方法 3 种。对象方法具有 self 参数;类方法使用修饰器@classmethod,具有 cls 参数;静态方法使用修饰器@staticmethod,不需要参数。

【例 8-7】 Python 三种方法举例。

```
class MyClass:
    def method(self):        #对象方法
        print("method")
    @staticmethod            #静态方法
    def staticMethod():
```

```
        print("static method")
    @classmethod            #类方法
    def classMethod(cls):
        print("class method")
```

8.4.1　对象方法

对象方法分为公有方法和私有方法两种。如果在方法名前面加了两个下划线"_ _",表示该方法是私有方法,否则为公有方法。对象方法与普通函数只有一个特别的区别,必须有一个额外的第一个参数名称(self),self 等同于 C++ 语言的 this 指针,用于指向对象本身,当对象调用该方法时,Python 就将对象作为第一个参数传递给 self。

(1) 公有方法

公有方法通过对象名调用。

【例 8-8】 公有方法举例。

```
class Person:               #声明 class 类
    def sayHi(self):        #公有方法
        print 'Hello, how are you? '

p=Person()                  #创建对象 p
p.sayHi()
输出
Hello, how are you?
```

(2) 私有方法

私有方法不能通过对象名调用,只能在对象的公有方法中通过 self 调用。

【例 8-9】 私有方法举例。

```
class Person:
    def __sayHi(self):      #私有方法
        print 'Hello, how are you? '
    def output(self):
        self.__sayHi()      #只能在对象的公有方法中通过 self 调用

p=Person()                  #创建对象 p
p.__sayHi()                 #错误,不能通过对象名调用
p.output()

输出
Hello, how are you?
```

8.4.2　类方法

类方法属于类,通过 Python 的修饰器@classmethod 实现,类方法只能通过类名调

用，具有 cls 参数。

【例 8-10】 类方法举例。

```
class MyClass:
    @classmethod
    def classMethod(cls):
        print("class method")
MyClass.classMethod()
输出
class method
```

8.4.3 静态方法

静态方法属于类，通过 Python 的修饰器@staticmethod 实现，静态方法只能通过类名调用，静态方法中不能访问属于对象的成员，只能访问属于类的成员。

【例 8-11】 静态方法举例。

```
class Fruit:
    price=0
    @staticmethod
    def getPrice():          #定义静态方法 getPrice
        return Fruit.price
    @staticmethod
    def setPrice(p):         #定义静态方法 setPrice
        Fruit.price=p
#主程序
print(Fruit.getPrice())
Fruit.setPrice(9)
print(Fruit.getPrice())
```

程序运行结果：

```
0
9
```

8.5 构造函数与析构函数

一般来说，对象的生命周期从构造函数开始，以析构函数结束。在创建类的实例时，调用构造函数为其分配内存。当类的实例生命周期结束前，调用析构函数释放占有的内存空间。

8.5.1 构造函数

_ _init_ _方法作为 Python 类的一种特殊方法，方法名的开始和结束都是双下划线，

该方法称为构造函数，用来为属性设置初值，每次创建类的实例时，构造函数都会自动执行。

Python 定义构造函数的语法格式为

```
def  __init__():
    函数体
```

【例 8-12】 构造函数举例。

```
class Complex:
    def __init__(self, realpart, imagpart):
        self.r=realpart
        self.i=imagpart
x=Complex(3.0, -4.5)
print x.r, x.i
```

【例 8-13】 构造函数举例。

```
class Person:
    def __init__(self, name):
        self.name=name
    def sayHi(self):
        print 'Hello, my name is', self.name
p=Person("zhou")
p.sayHi()
输出

Hello, my name is zhou
```

注意： __init__方法可选，但是子类一旦定义，就必须显示调用父类的__init__方法。

8.5.2 析构函数

析构函数是__del__，用来释放对象占用的资源，在 Python 收回对象空间之前自动执行，自动完成内存清理工作，又称为垃圾收集器。

【例 8-14】 析构函数举例。

```
class Car:
        def __init__(self, n):
        self.num=n
        print 'number is', self.num, 'object is born...'
    def __del__(self):
        print 'number is', self.num, 'object is dead...'
car1=Car(1)
car2=Car(2)
del car1
```

```
del car2
```

运行结果如图 8.4 所示。

```
C:\WINDOWS\system32\cmd.exe - PAUSE
number is 1 object is born...
number is 2 object is born...
number is 1 object is dead...
number is 2 object is dead...
请按任意键继续. . .
```

图 8.4　例 8-14 程序运行结果

8.6　继承性

教师和学生有一些共同属性，如姓名、年龄和地址等，也有各自专有的属性，例如教师的薪水、课程和假期，学生的成绩。如果教师和学生是两个独立的类，要增加一个新的共有属性，就意味着要在这两个独立的类中都增加，这样会很烦琐。较好的方法是创建一个共同的类(假设称为 SchoolMember)，教师和学生的类继承这个共同的类，将新的共有属性增加到 SchoolMember 类中即可。

在 OO 程序设计中，继承性是通过派生类和基类实现的，基类又称为父类或超类(Base class，Super class)，而派生类又称为子类(Subclass)。

Python 继承语法如下所示：

```
class SubClassName (ParentClass1[, ParentClass2, ...]):
    class_suite
```

说明：

(1) 基类只是简单地列在类名后面的小括号里。

(2) Python 支持多重继承，只须在类名后面的小括号中列出多个基类名，以逗号分隔。

(3) 基类的构造(_ _init_ _()方法)不会被自动调用，必须在派生类中显示调用父类的_ _init_ _方法。

(4) 调用基类的方法，需要加上基类的类名作为前缀，带上 self 参数变量，而在类中调用普通函数时并不需要带上 self 参数。

【例 8-15】 继承性举例。

```
class SchoolMember:
    '''Represents any school member.'''
    def __init__(self, name, age):
        self.name=name
        self.age=age
        print '(Initialized SchoolMember: %s)' %self.name
```

```
    def tell(self):
        '''Tell my details.'''
        print 'Name:"%s" Age:"%s"' %(self.name, self.age),

class Teacher(SchoolMember):
    '''Represents a teacher.'''
    def __init__(self, name, age, salary):
        SchoolMember.__init__(self, name, age)
        self.salary=salary
        print '(Initialized Teacher: %s)' %self.name

    def tell(self):
        SchoolMember.tell(self)
        print 'Salary: "%d"' %self.salary

class Student(SchoolMember):
    '''Represents a student.'''
    def __init__(self, name, age, marks):
        SchoolMember.__init__(self, name, age)
#子类的构造函数必须显式调用基类的构造函数
        self.marks=marks
        print '(Initialized Student: %s)' %self.name

    def tell(self):
        SchoolMember.tell(self)
        print 'Marks: "%d"' %self.marks

t=Teacher('Mrs. Shrividya', 40, 30000)
s=Student('Swaroop', 22, 75)
members=[t, s]
for member in members:
    member.tell()           #works for both Teachers and Students
```

程序运行结果如图 8.5 所示。

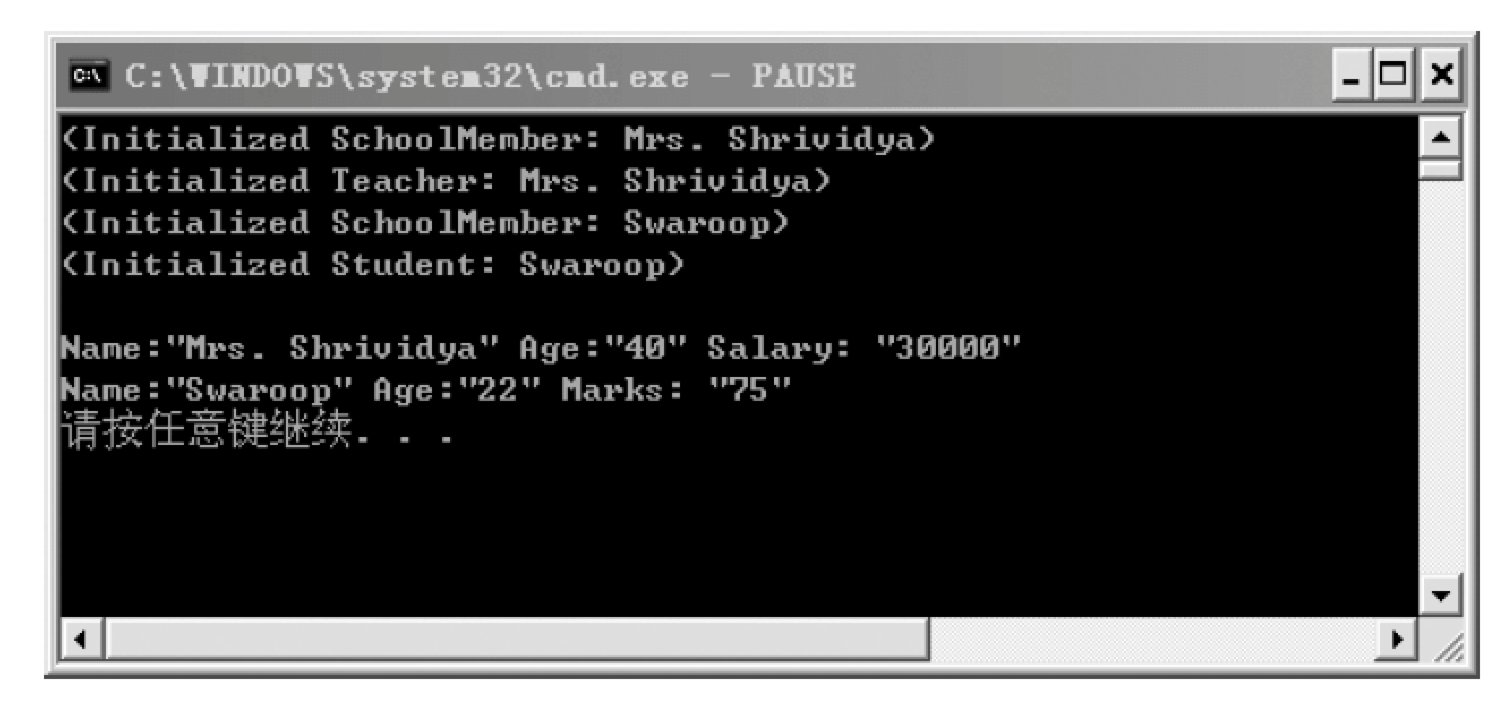


图 8.5　例 8-15 程序运行结果

【例 8-16】 多继承举例。

```
class A:
    name="A"
    __num=1
    def show(self):
        print self.name
        print self.__num
    def setnum(self,num):
        self.__num=num

class B:
    nameb='B'
    __numb=2
    def show(self):
        print self.nameb
        print self.__numb
    def setname(self,name):
        self.__name=name

class C(A,B):
    def showall(self):
        print self.name
        print self.nameb
a=A()
a.show()
b=B()
b.show()
c=C()
c.showall()
```

程序运行结果如图 8.6 所示。



图 8-6 例 8-16 程序运行结果

8.7　多态性

Python 通过方法重载和运算符重载两种方式实现多态性。

8.7.1　方法重载

方法重载实际上就是在子类中使用与父类完全相同的方法名，从而重载父类的方法。

【例 8-17】 方法重载。

```
class Parent:
    def myMethod(self):
        print 'Calling parent method'
class Child(Parent):
    def myMethod(self):
        print 'Calling child method'

c=Child()
p=Parent()
c.myMethod()          #子类调用重载方法
p.myMethod()          #父类的方法
```

程序运行结果如图 8.7 所示。



图 8-7　例 8-18 程序运行结果

8.7.2　运算符重载

在 C# 中，运算符重载是通过使用关键字 operator 实现。Python 的运算符重载方式较为简单，Python 规定每一个类都默认内置了所有可能的运算符方法，只要重写这个方法，就可以实现针对该运算符的重载。

【例 8-18】 运算符重载举例。

```
class OpeClass():
    def __init__(self):
        self.a=11
        self.b=22
    def __add__(self,x):
        return self.a+self.b
    def __sub__(self,x):
```

```
        return self.a- self.b

a=OpeClass()
b=OpeClass()
print a+b
print a-b
```

输出

```
33
-11
```

8.8 习题

1. 如何实现类的继承？

2. 如何在类中定义属性？

3. 如何实现方法的重载？

4. 编写人员类(Person)，该类具有姓名(Name)、年龄(Age)、性别(Sex)等域。然后通过对Person类的继承得到一个教师类(Teacher)，该类能够存放教师的职称、学历、工资、奖金等信息，并能计算出总收入(工资+奖金)，要求对该类构造函数进行重载。

第 9 章

文　件

文件是指在各种存储介质上永久存储的数据集合，如 Word 以.doc 形式存在，将其保存在磁盘上就是磁盘文件，输出到打印机上就是一个打印机文件。本章首先介绍文件的相关概念和字符编码，就 Python 的文件操作进行说明，讲解存储器的概念，最后讲述几个与文件有关的模块。

9.1　文件概念

9.1.1　字符编码

字符编码作为计算机技术的基石，常见的字符编码有 ASCII，UTF-8，Unicode 和 GB 2312 等。

（1）ASCII 编码

在计算机内部，所有的信息最终都表示为二进制的字符串。每一个二进制位（bit）有 0 和 1 两种状态，8 个二进制位称为一个字节（byte），可以组合出 256 种状态。20 世纪 60 年代，美国制定了 ASCII 编码（American Standard Code for Information Interchange）。ASCII 编码使用指定的 7 位或 8 位二进制数组合来表示 128 或 256 种可能的字符，将英语字符与二进制之间的关系进行了规定，可以表示 26 个大写和小写字母，数字 0 到 9、标点符号，以及在美式英语中使用的特殊控制字符。

（2）GB 2312 编码

GB 2312 编码是第一个汉字编码国家标准，由中国国家标准总局 1980 年发布，1981 年 5 月 1 日开始使用。GB 2312 编码共收录汉字 6763 个，其中一级汉字 3755 个，二级汉字 3008 个。同时，GB 2312 编码收录了包括拉丁字母、希腊字母、日文平假名及片假名字母、俄语西里尔字母在内的 682 个全角字符。

（3）Unicode 编码

Unicode 编码将世界上的每一个符号进行独一无二的编码，解决了乱码问题。Unicode 的学名是 Universal Multiple-Octet Coded Character Set，简称为 UCS。Unicode 又称为抽象编码，它只是一个符号集，并没有规定这些符号的二进制代码应该如何存储和如何传输，UCS 的传输由 UTF-8 或 UTF-16 规范规定。

(4) UTF 编码

浏览网页的源码，发现会有类似<meta charset="UTF-8" />的信息，表示该网页为 UTF-8 编码。UTF-8 作为 unicode 编码的实现方式之一，以 8 位(1 字节)表示英语，以 24 位(3 字节)表示中文及其他语言。

9.1.2 文件分类

根据文件编码方式的不同，Python 将文件分为文本文件和二进制文件。

(1) 文本文件。文本文件又称为 ASCII 文件，是由 ASCII 编码字符组成并且不带任何格式的文件，通常使用字处理软件(如 Windows 的记事本)编辑。文本文件的读取必须从文件的头部开始，一次全部读出，不能只读取中间的一部分数据，不可以跳跃式访问。文本文件的每一行文本相当于一条记录，每条记录可长可短，记录之间使用“换行符”进行分隔，不能同时进行读、写操作。文本文件的优点是使用方便，占用内存资源较少，但其访问速度较慢，并且不易维护。

(2) 二进制文件。二进制文件作为最原始的文件类型，直接把二进制码存放在文件中，以字节为单位访问数据，不能用字处理软件进行编辑。二进制文件允许程序按所需的任何方式组织和访问数据，也允许对文件中各字节数据进行存取和访问。

除此之外，根据存储数据的性质将文件分为程序文件和数据文件；根据文件的流向分为输入文件和输出文件；根据文件的存储介质分为磁盘文件、磁带文件等。

9.2 文件打开和关闭

文件访问主要是对文件进行读、写操作。读文件是将文件中的数据读入计算机内存，即向计算机输入数据。写文件将计算机内存中的数据写入文件中。虽然文件的记录组织方式和数据编码格式不同，但是 Python 操作文件的基本步骤相同，都经过以下三个步骤完成，如图 9.1 所示。

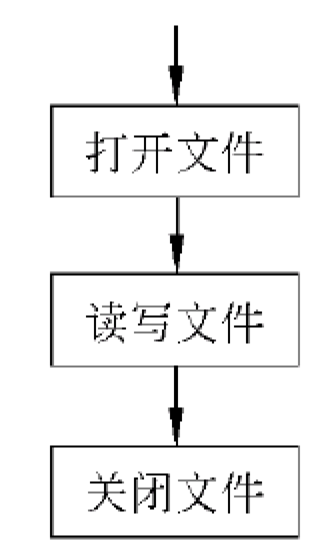


图 9.1 文件操作的三个步骤

(1) 打开文件，如果文件不存在应先创建文件。

(2) 当文件打开后，进行读或写操作。

(3) 文件操作完毕，关闭文件。

Python 操作文件的语法如下所示：

```
文件变量名=open(文件名[, 打开方式])
…         #处理
文件变量名.close()
```

Python 的内置函数 open()用来打开磁盘上的文件信息。Close()用于关闭文件。

说明：

(1) 文件名指定了被打开的文件名称。

（2）打开方式指定了打开文件后的处理方式，见表9.1。

表9.1 Python打开文件方式

方 式	意 义	存 在	不存在
"r"	为只读打开文本文件	打开	返空指针
"w"	为只写打开新文本文件	打开删空	新建打开
"a"	为追加打开文本文件	打开	新建打开
"rb"	为只读打开二进制文件	打开	返空指针
"wb"	为只写打开新二进制文件	打开删空	新建打开
"ab"	为追加打开二进制文件	打开	新建打开
"r+"	为读打开文本文件可写	打开	返空指针
"w+"	为写打开新文本文件可读	打开删空	新建打开
"a+"	为追加打开文本文件可读	打开	新建打开
"rb+"	为读打开二进制文件可写	打开	返空指针
"wb+"	为写打开新二进制文件可读	打开删空	新建打开
"ab+"	为追加打开二进制文件可读	打开	新建打开

说明如下：

（1）访问方式中，r表示只读，w表示只写，a表示在文件末尾追加，+表示增加其他访问方式，b表示二进制文件，默认表示ASCII码文件（文本文件）。

（2）所有带"r"的方式打开的文件都应该存在。

（3）所有带"w"和"a"的方式，被打开的文件不存在，则建新文件并打开，若该文件已存在，则删空重写。

（4）所有带"a"的方式，打开后文件位置指针指向文件末尾，其他方式打开后位置指针指向文件开始。

（5）所有带"+"方式打开的文件，既可读，也可写。

（6）读文本文件时，系统将回车换行符换为单个换行符，写时又把换行符换为回车换行符。用二进制文件时，不进行这种转换，内存中数据与输出文件完全一致。

文件读写时可能产生IOError错误，从而导致无法正确地关闭文件，因此，往往使用异常操作（try…finally）保证文件安全。

【例9-1】 文件打开关闭举例。

```
try:
    f=open('d:\poem.txt', 'r')
    print f.read()
finally:
    if f:
        f.close()
```

9.3 文件操作

9.3.1 写操作

Python 提供了 write 函数用于将数据写入到文件中。

【例 9-2】 write 举例。

```
f=open('d:\poem.txt', 'w')          #open for 'w'riting
poem='''\
Programming is fun
When the work is done
if you wanna make your work also fun:
      use Python!
'''
f.write(poem)                       #write text to file
f.close()                           #close the file
```

【解析】 程序运行后，在 D:盘下产生一个 poem.txt 文件。

9.3.2 读操作

当使用 open()以“r”读模式打开文件后，Python 提供了三个方法用于文本文件的读取，即 read()、readline()和 readlines()。

(1) read 方法

read 方法用于一次性将文件里的内容全部读取，也可以指定每次读多少个字节，如 read(5)就是从文件开始读取 5 个字节。

【例 9-3】 read()方法的使用。

```
f=open('d:\poem.txt',"r")
re=f.read()
print re
f.close()
```

输出结果如图 9.2 所示。

```
C:\WINDOWS\system32\cmd.exe - PAUSE
Programming is fun
When the work is done
if you wanna make your work also fun:
        use Python!

请按任意键继续. . .
```

图 9.2　例 9-3 程序运行结果

【解析】 read 函数将 d:\poem.txt 文件的内容读出，输出到屏幕上。

(2) readline 方法

readline 方法用于一行行地读出并显示文件内容。如果读到文件末尾,就返回一个空字符串。Readline 方法读取文件的流程如图 9.3 所示。

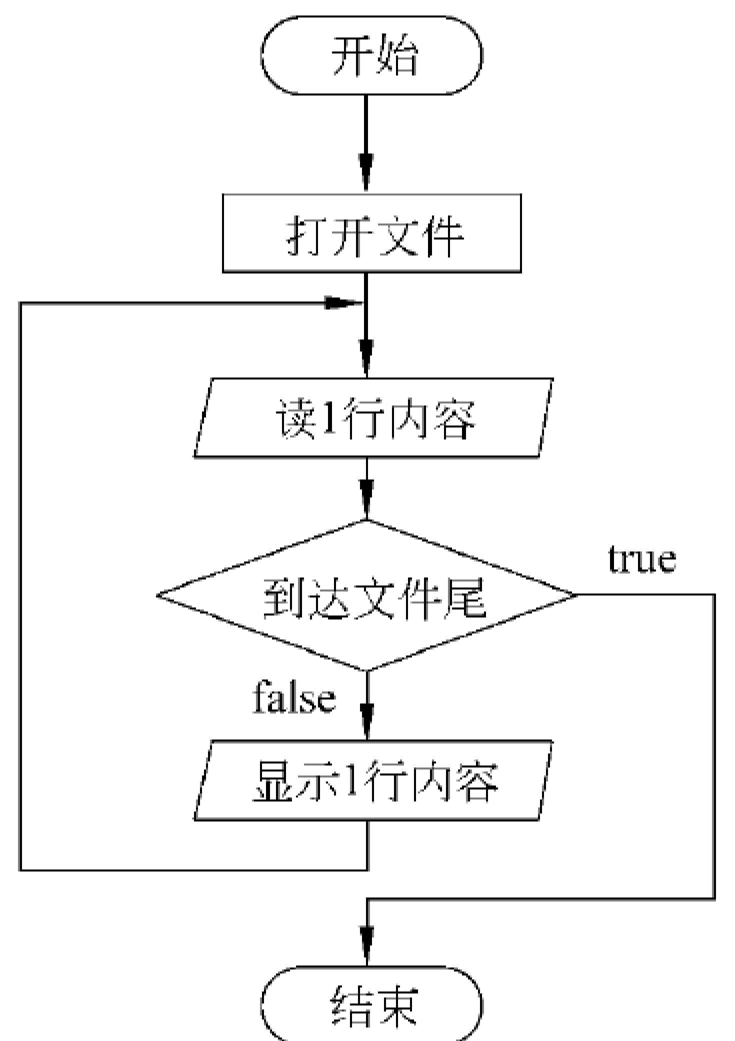


图 9.3 用 readline 方法读取文件

【例 9-4】 readline()方法的使用。

```
f=open('d:\poem.txt',"r")
line=f.readline()
while line:
    print line,
    line=f.readline()
f.close()
```

(3) readlines 方法

readlines 方法自动将文件内容分析成一个行的列表,该列表可以由 Python 的 for…in…结构进行处理。readline()每次只读取一行,通常比 readlines()慢得多。

【例 9-5】 readlines()方法的使用。

```
f=open('d:\poem.txt','r')
for line in f.readlines():
    print line
f.close()
```

9.3.3 文件指针

为了在文件的任何位置读写内容,Python 提供了 seek()方法用于移动文件指针。具体语法如下所示:

(1) seek(n),其中 n>=0,seek(0)表示文件指针移到文件头;n>0 时,表示移动到文件头之后的位置。

(2) seek(0,2),表示把文件指针移到文件尾,当在文件尾部追加新内容时需要使用。

【例 9-6】 文件指针举例。

```
f=open('d:\poem.txt',"r")
re=f.read()
print re
f.close()

f=open('d:\poem.txt',"r+")
f.seek(0)               #文件指针移到文件头
f.write('hello')        #增加新内容,将覆盖原文件相同位置的字符
f.seek(9)               #文件指针移到第 9 个字符
f.write('OK')           #增加新内容,将覆盖原文件相同位置的字符
```

```
f.seek(0, 2)            #文件指针移到文件尾
f.write('bye')          #增加新内容
f.close()

f=open('d:\poem.txt',"r")
re=f.read()
print re
f.close()
```

运行结果如图 9.4 所示。

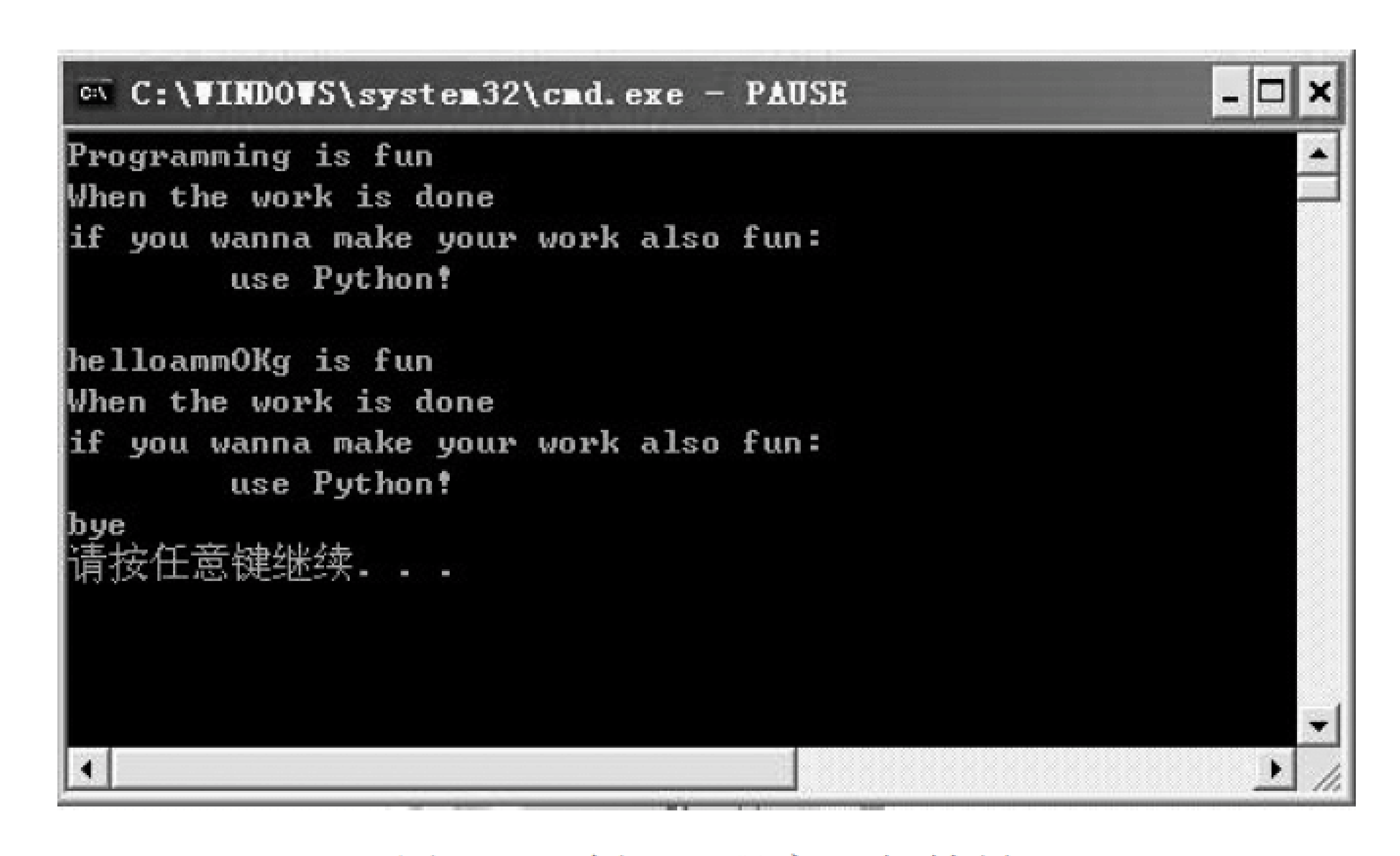


图 9.4 例 9-6 程序运行结果

注意：不论是二进制文件还是文本文件，指针的相对位置的计算都是以字节为单位。

9.4 存储器

Python 提供 pickle 模块和 cPickle 模块用于在文件中存储和读取数据。pickle 模块的 dump 函数把文件对象转换为字符串存储到文件中，这个过程称为存储。pickle 模块的 load 函数用于从文件中取回对象，这个过程称为取存储。

【例 9-7】 文件的存储与取存储。

```
import cPickle as p                #import pickle as p

shoplistfile="d:\shoplist.txt"
shoplist=['apple', 'banana', 'orange']

f=file(shoplistfile, 'w')          #Write to the file
p.dump(shoplist, f)                #dump the object to a file
f.close()

del shoplist                       #remove the shoplist

f=file(shoplistfile)
storedlist=p.load(f)               #Read back from the storage
print storedlist
```

9.5 与文件相关的模块

9.5.1 os 模块

Python 内置的 os 模块直接调用操作系统提供的接口函数，用于对目录和文件进行操作。当导入 os 模块，应该使用“import os”，而不是“from os import * ”，用于保证 os .open不会随操作系统的不同而覆盖 open()内置函数。

【例 9-8】 os 模块。

```
>>> import os
>>> os.name                                      #操作系统名字
'nt'
#nt 说明操作系统是 Windows;posix 说明操作系统是 Linux,UNIX 或 Mac OS X
>>> os.rename("test1.txt", "test2.txt")          #重命名文件 test1.txt 到 test2.txt
>>> os.remove("text2.txt")                       #删除已经存在的文件 test2.txt
>>> os.listdir(path)                             #列出目录下的文件
>>> os.getcwd()                                  #获取当前工作目录
>>> os.makedirs(r"c:\python\test")               #创建多级目录
>>> os.removedirs(r"c:\python")                  #删除多个目录
>>> os.mkdir('/Users/zhou/testdir')              #mkdir()方法用于创建一个目录
>>> os.chdir('/Users/zhou/testdir')              #chdir()方法改变一个目录
#rmdir()方法用于删掉目录,在删除目录之前,其所有内容应该先被清除
>>> os.rmdir('/Users/zhou/testdir')
```

9.5.2 os. path 模块

os. path 模块主要用于文件的属性获取，以下是该模块的几种常用方法。

【例 9-9】 os. path 模块。

```
>>> import os.path
#查看当前目录的绝对路径
>>> os.path.abspath('.')
'/Users/zhou'
#目录下创建子目录
>>> os.path.join('/Users/zhou', 'testdir')
'/Users/zhou/testdir'
#分离文件名
>>> os.path.split(r"c:\python\hello.py") --> ("c:\\python","hello.py")
#分离扩展名
>>> os.path.splitext(r"c:\python\hello.py")--> ("c:\\python\\hello",".py")
    #获取路径名
>>> os.path.dirname(r"c:\python\hello.py")
"c:\\python"
    #获取文件名
>>> os.path.basename(r"r:\python\hello.py")
```

```
"hello.py"
    #判断文件是否存在
>>>os.path.exists(r"c:\python\hello.py")-->True
    #判断是否是绝对路径
>>>os.path.isabs(r".\python\")
False
#判断是否是目录
>>>os.path.isdir(r"c:\python")
True
    #判断是否是文件
>>>os.path.isfile(r"c:\python\hello.py")
True
    #判断是否是链接文件
>>>os.path.islink(r"c:\python\hello.py")
False
    #获取文件大小
>>>os.path.getsize(filename)
```

9.5.3 shutil 模块

针对日常的文件和目录管理任务，shutil 模块提供了一个更加易于使用的高级接口。

【例 9-10】 shutil 模块。

```
>>>import shutil
>>>dir(shutil)
#复制单个文件
>>>shultil.copy(oldfile,newfle)
#复制整个目录树
>>>shultil.copytree(r".\setup",r".\backup")
#删除整个目录树
>>>shultil.rmtree(r".\backup")
```

运行结果如图 9.5 所示。

```
C:\WINDOWS\system32\cmd.exe - python
Microsoft Windows XP [版本 5.1.2600]
(C) 版权所有 1985-2001 Microsoft Corp.

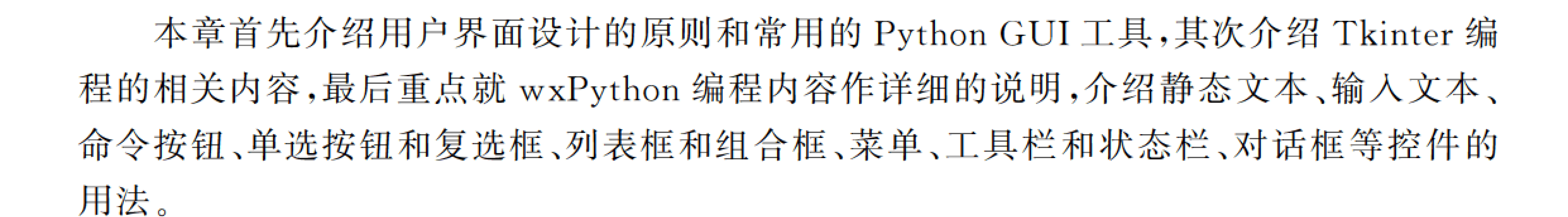
C:\Documents and Settings\Administrator>python
Python 2.7.3 (default, Apr 10 2012, 23:31:26) [MSC v.1500 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import shutil
>>> dir(shutil)
['Error', 'ExecError', 'SpecialFileError', '_ARCHIVE_FORMATS', '__all__', '__builtins__', '__doc__',
 '__file__', '__name__', '__package__', '_basename', '_call_external_zip', '_destinsrc', '_get_gid',
 '_get_uid', '_make_tarball', '_make_zipfile', '_samefile', 'abspath', 'collections', 'copy', 'copy2
', 'copyfile', 'copyfileobj', 'copymode', 'copystat', 'copytree', 'errno', 'fnmatch', 'get_archive_f
ormats', 'getgrnam', 'getpwnam', 'ignore_patterns', 'make_archive', 'move', 'os', 'register_archive_
format', 'rmtree', 'stat', 'sys', 'unregister_archive_format']
>>> _
```

图 9-5 例 9-10 程序运行结果

9.6 习题

1. 什么是文件？文件分几类？各是什么？
2. Python 提供了几种文件访问的方法？分别是什么？
3. 文本文件、二进制文件之间有什么异同点？
4. 文本文件的读写函数有哪些？分别解释其功能和参数。
5. 二进制文件的读写函数有哪些？分别解释其功能和参数。
6. 存储器是什么？如何进行文件的读写操作？

第10章 用户界面设计



本章首先介绍用户界面设计的原则和常用的 Python GUI 工具，其次介绍 Tkinter 编程的相关内容，最后重点就 wxPython 编程内容作详细的说明，介绍静态文本、输入文本、命令按钮、单选按钮和复选框、列表框和组合框、菜单、工具栏和状态栏、对话框等控件的用法。

10.1 概述

用户界面作为程序最重要的组成部分，主要负责用户与应用程序之间的交互。用户界面设计往往需要考虑许多问题，例如，使用单文档界面还是多文档界面，菜单是否嵌套，工具栏是否有必要重复菜单的功能，界面提供哪些帮助信息，等等。

10.1.1 界面设计原则

界面设计一般具有如下原则：

(1) 界面具有一致性。例如，在菜单和联机帮助中必须使用相同的术语；对话框必须具有相同的风格等。

(2) 常用操作设置快捷键。常用操作的使用频度大，应该减少操作序列的长度。例如，文件的常用操作(如打开、存盘、另存为等)设置快捷键将会提高工作效率。

(3) 提供简单错误处理。系统对于错误能检测，并提供快速简单的处理办法。

(4) 提供信息反馈。对常用操作和简单操作的反馈可以不做要求，但是对不常用操作和至关重要的操作，应提供信息的反馈。

(5) 操作可逆。可逆的动作可以是单个的操作，也可以是相对独立的操作序列。

(6) 联机帮助。对于不熟练的用户，良好的联机帮助具有非常重要的作用。

10.1.2 常用 GUI 工具

许多优秀的 GUI 工具可以集成在 Python 中，常用的 GUI 如下。

(1) Tkinter：Tkinter 模块(“Tk 接口”)是 Python 的标准 Tk GUI 工具包的接口，Tk 和 Tkinter 可以在大多数的 UNIX 平台上使用，也可应用于 Windows 和 Macintosh 系统。

（2）wxPython：wxPython 是一款开源软件，作为 Python 语言的优秀 GUI 图形库，能快速方便地创建完整、功能键全的 GUI 用户界面。wxPython 下载网址为 http://wxpython.org/download.php。

（3）Jython：Jython 与 Java 无缝集成，直接使用 Java 中的 Swing，AWT 或者 SWT。

10.2　Tkinter 编程

10.2.1　Tkinter 简介

Tkinter 是 Python 的标准 GUI 库，Python 自带的 IDLE 就是采用 Tkinter 开发实现的。Tkinter 内置在 python 中，简单实用。

导入 Tkinter 模块一般采用如下 3 种方式：

- import Tkinter　导入 Tkinter 模块；
- import Tkinter as tk　导入 Tkinter 为 tk；
- from tkinter import *　导入 Tkinter 所有内容。

Tkinter 提供各种控件，如按钮、标签和文本框等，如表 10.1 所示。

表 10.1　Tkinter 控件

控　件	描　述
Button	按钮控件：在程序中显示按钮
Canvas	画布控件：显示图形元素（如线条或文本）
Checkbutton	多选框控件：在程序中提供多项选择框
Entry	输入控件：显示简单的文本内容
Frame	框架控件：在屏幕上显示一个矩形区域，多用作容器
Label	标签控件：可以显示文本和位图
Listbox	列表框控件：在 Listbox 窗口小部件是用来显示一个字符串列表给用户
Menubutton	菜单按钮控件：显示菜单项
Menu	菜单控件：显示菜单栏、下拉菜单和弹出菜单
Message	消息控件：显示多行文本，与 label 比较类似
Radiobutton	单选按钮控件：显示一个单选的按钮状态
Scale	范围控件：显示一个数值刻度，为输出限定范围的数字区间
Scrollbar	滚动条控件：当内容超过可视化区域时使用，如列表框
Text	文本控件：显示多行文本
Toplevel	容器控件：提供一个单独的对话框，与 Frame 类似
Spinbox	输入控件：与 Entry 类似，但是可以指定输入范围值
PanedWindow	PanedWindow：一个窗口布局管理的插件，可以包含一个或者多个子控件
LabelFrame	labelframe：一个简单的容器控件。常用于复杂的窗口布局
tkMessageBox	tkMessageBox：显示应用程序的消息框

10.2.2 实例讲解

【例 10-1】 最简单的 tk 举例。

```
import Tkinter              #导入 Tkinter 模块
top=Tkinter.Tk()
top.mainloop()              #进入消息循环
```

运行结果如图 10.1 所示。



图 10.1 例 10-1 程序运行结果

【例 10-2】 编写 GUI 版本的“Hello，world!”。

实现此功能共有如下三步：

第一步：导入 Tkinter 包的所有内容。

```
from Tkinter import *
```

第二步：从 Frame 类派生 Application 类，Frame 是所有功能块(Widget)的父容器。

```
class Application(Frame):
    def __init__(self, master=None):
        Frame.__init__(self, master)
        self.pack()
        self.createWidgets()

    def createWidgets(self):
        self.helloLabel=Label(self, text='Hello, world!')
        self.helloLabel.pack()
        self.quitButton=Button(self, text='Quit', command=self.quit)
        self.quitButton.pack()
```

【解析】 GUI 中的每个 Button 和 Label 都是一个 Widget。Frame 是可以容纳其他 Widget 的 Widget。

pack()方法把 Widget 加入到父容器中实现布局。

createWidgets()方法中，创建一个 Label 和一个 Button，当单击 Button 时，触发 self.quit()方法退出程序。

第三步：实例化 Application，并启动消息循环。

```
app=Application()
#设置窗口标题：
app.master.title('Hello World')
#主消息循环：
app.mainloop()
```



图 10.2 例 10-2 程序运行结果之一

运行结果如图 10.2 所示，单击 Quit 按钮或者窗口的 x 结束程序。

对例 10-2 增加如下功能：增加文本框实现用户

输入，单击按钮弹出消息对话框。运行结果如图 10.3 所示。

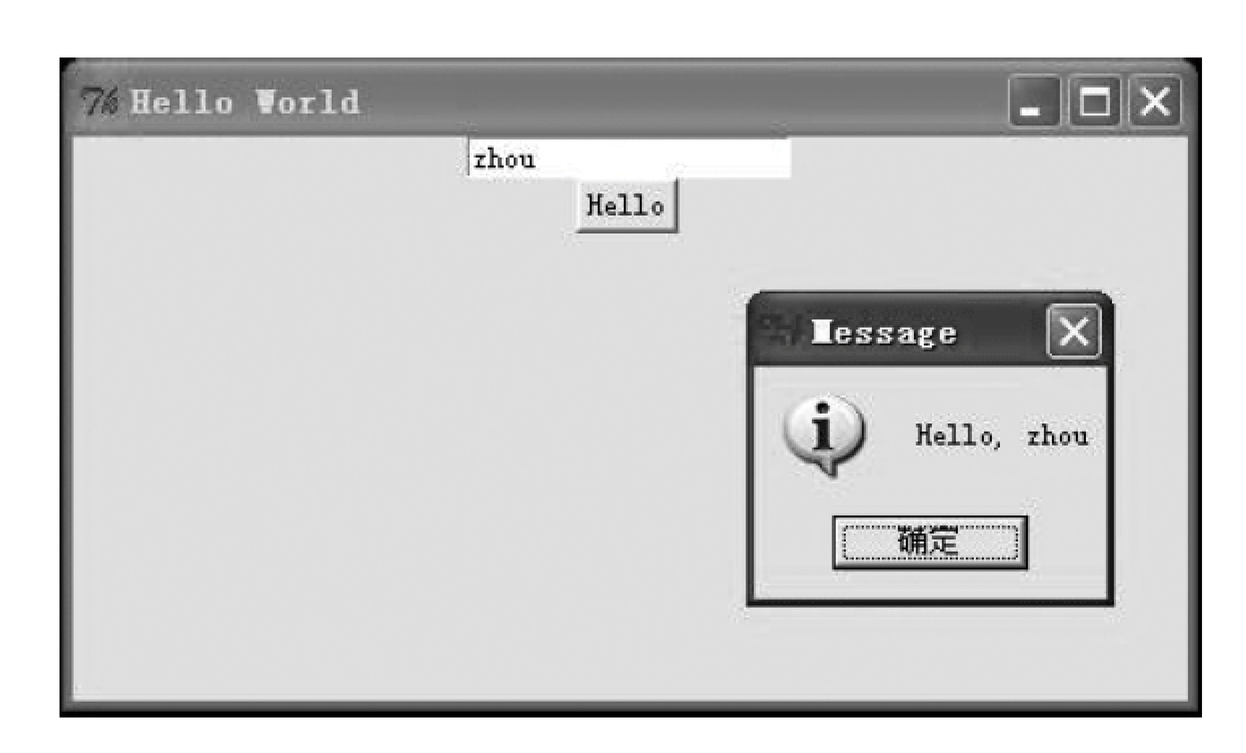


图 10.3　例 10-2 程序运行结果之二

```
from Tkinter import *
import tkMessageBox

class Application(Frame):
    def __init__(self, master=None):
        Frame.__init__(self, master)
        self.pack()
        self.createWidgets()

    def createWidgets(self):
        self.nameInput=Entry(self)
        self.nameInput.pack()
        self.alertButton=Button(self, text='Hello',command=self.hello)
        self.alertButton.pack()

    def hello(self):
        name=self.nameInput.get()
        tkMessageBox.showinfo('Message', 'Hello, %s' %  name)
```

用户单击按钮触发 hello()，通过 self.nameInput.get()获得用户输入的文本后，使用 tkMessageBox.showinfo()弹出消息对话框。

10.3　wxPython 编程

10.3.1　wxPython 简介

Python 内置的 Tkinter 满足用户界面设计的基本要求，但其功能较为简单，对于较为复杂的 GUI 程序一般采用 wxPython 进行设计，wxPython 下载网址为 http://wxpython.org/，如图 10.4 所示。

本书所用 Python 版本为 32 位 Python2.7.3，下载 wxPython3.0-win32-py27 文件，双击安装，默认路径为 C:\Python27\Lib\site-packages。



图 10.4　wxPython 下载网址

10.3.2　wxPython 开发流程

wxPython 程序包括应用程序(Application)对象和框架(Frame)对象。框架对象作为界面的容器,其功能相当于画布。应用程序用于设计程序的总体框架,一般默认为单文档界面(Single Document Interface,SDI)。

wxPython 开发流程包括如下三大步骤:

步骤 1:导入必需的 wxPython 包。

步骤 2:建立框架类:框架类的父类为 wx.Frame,在框架类的构造函数中必须调用父类的构造函数。

步骤 3:主程序通常做 4 件事。

① 建立应用程序对象;

② 建立框架类对象;

③ 显示框架;

④ 建立事件循环。

10.3.3　Frame 创建与使用

wx.Frame 作为程序的运行界面,可以容纳其他构件,Python 语法格式如下:

```
wx. Frame ( wx. Window parent, id, string title, wx. Point pos = wx.
DefaultPosition, wx.Size size=wx.DefaultSize, style=wx.DEFAULT_FRAME_STYEL, string
name='frame')
```

参数说明:

parent:框架的父窗体。对于顶级窗体,该值为 None。

id:新窗体的 wxPython ID 号,用于事件和处理事件函数之间建立唯一的关联,一般

使用全局常量 wx. ID_ANY(其值为－1)。

title：窗体的标题。

pos：wx. Point 对象用于指定窗体的左上角位置。(0,0)是窗体的左上角位置。wx. DefaultPosition 是(－1,－1),表示窗体的初始位置。

size：wx. Size 对象用于指定窗体的尺寸。wx. DefaultSize 是(－1,－1),表示窗体的初始尺寸。

style：指定窗体类型的常量。

name：框架的名字,指定后可以使用它来寻找这个窗体。

注意：一般情况,wx. Frame 接受 parent,id 和 title 3 个参数,parent 必须赋值,其余都有默认值。

创建一个 wx. Frame 构件,它没有 parent,标识符是 100,标题是 Title,位置在(100,50),大小是 (100,100)：

```
frame=wx.Frame(None,-1,'Title',wx.Point(100,50),wx.Size(100,100))
```

省略了 pos 参数,明确提供 size 参数：

```
frame=wx.Frame(None,100,'Title',size=wx.Size(100,100))
```

Frame 有如下方法：

- SetTitle(string title)——设置窗口标题。只可用于框架和对话框。
- SetToolTip(wx. ToolTip tip)——为窗口添加提示。
- SetSize(wx. Size size)——设置窗口的尺寸。
- SetPosition(wx. Point pos)——设置窗口出现的位置。
- Show(show＝True)——显示或隐藏窗口。其中的参数可以为 true 或 false。
- Move(wx. Point pos)——将窗口移动到指定位置。
- SetCursor(wx. StockCursor id)——设置窗口的鼠标指针样式。

【例 10-3】　框架(Frame)创建与使用。

```
import wx                       #导入必需的 wxPython 包
class Frame(wx.Frame):
    def __init__(self):
        wx.Frame.__init__(self, None, -1, "the third GUI", size=(300, 300))
if __name__=='__main__':
    app=wx.App()                #建立应用程序对象
    frame=Frame()               #建立框架类对象
    frame.Show()                #显示框架
    app.MainLoop()
```

建立事件循环,程序控制权将转交给 wxPython,响应用户的鼠标和键盘事件。

程序运行结果如图 10.5 所示。



图 10.5　例 10-3 程序运行结果

【解析】　解释如下两点：

(1) __init__()

声明子类 Frame 声明了__init__()方法，必须显式调用基类的__init()__方法，代码如下所示：

```
class Frame(wx.Frame):
    def __init__(self):
        wx.Frame.__init__(self)
```

(2) if __name__ == '__main__':

程序由几个.py 文件组成，用于开启程序的文件称为 main，Python 通过__name__这个特殊的变量调用开启文件，其调用代码为(if __name__ == '__main__':)。

10.4 控件

wxPython 包括多种基本控件，如静态文本、输入文本、命令按钮、微调控制框、滑块、复选框、单选按钮、列表框、组合框等。当将控件放到框架上时，会生成与框架相同大小的 wx.Panel(面板)，面板使框架上所有的控件与工具栏和状态栏分开，使得控件可以通过 Tab 键进行遍历访问。

10.4.1 静态文本

静态文本与 Visual Basic 语言中的标签功能相似，用于显示文本内容，但不能编辑。wxPython 使用类 wx.StaticText 来完成。

【例 10-4】 静态文本举例。

```
import wx
class StaticTextFrame(wx.Frame):
    def __init__(self):
        wx.Frame.__init__(self, None, -1, 'Static Text Example',size=(400, 300))
        panel=wx.Panel(self, -1)          #面板
        #静态文本
        wx.StaticText(panel, -1, "This is an example of static text", (100, 10))
        #指定了前景色和背景色的静态文本
        rev=wx.StaticText(panel, -1, "Static Text With Reversed Colors", (100, 30))
            rev.SetForegroundColour('white')
            rev.SetBackgroundColour('black')
            #指定居中对齐的静态文本
            center=wx.StaticText(panel, -1, "align center", (100, 50),(160,
            -1), wx.ALIGN_CENTER)
            center.SetForegroundColour('white')
            center.SetBackgroundColour('black')
            #指定右对齐的静态文本
            right=wx.StaticText(panel, -1, "align right", (100, 70), (160,
            -1), wx.ALIGN_RIGHT)
```

```
            right.SetForegroundColour('white')
            right.SetBackgroundColour('black')
            #指定新字体的静态文本
            str="You can also change the font."
            text=wx.StaticText(panel, -1, str, (20, 100))
            font=wx.Font(20, wx.DECORATIVE, wx.ITALIC, wx.NORMAL)
            text.SetFont(font)
            #显示多行文本
            wx.StaticText(panel, -1, "Your text\ncan be split\nover multiple lines\n\
            neven blank ones", (20,150))
            #显示对齐的多行文本
            wx.StaticText(panel, -1, "Multi-line text\ncan also\nbe right aligned\n\
            neven with a blank", (220,150), style=wx.ALIGN_RIGHT)
if __name__=='__main__':
    app=wx.App()
    frame=StaticTextFrame()
    frame.Show()
    app.MainLoop()
```

程序运行结果如图 10.6 所示。

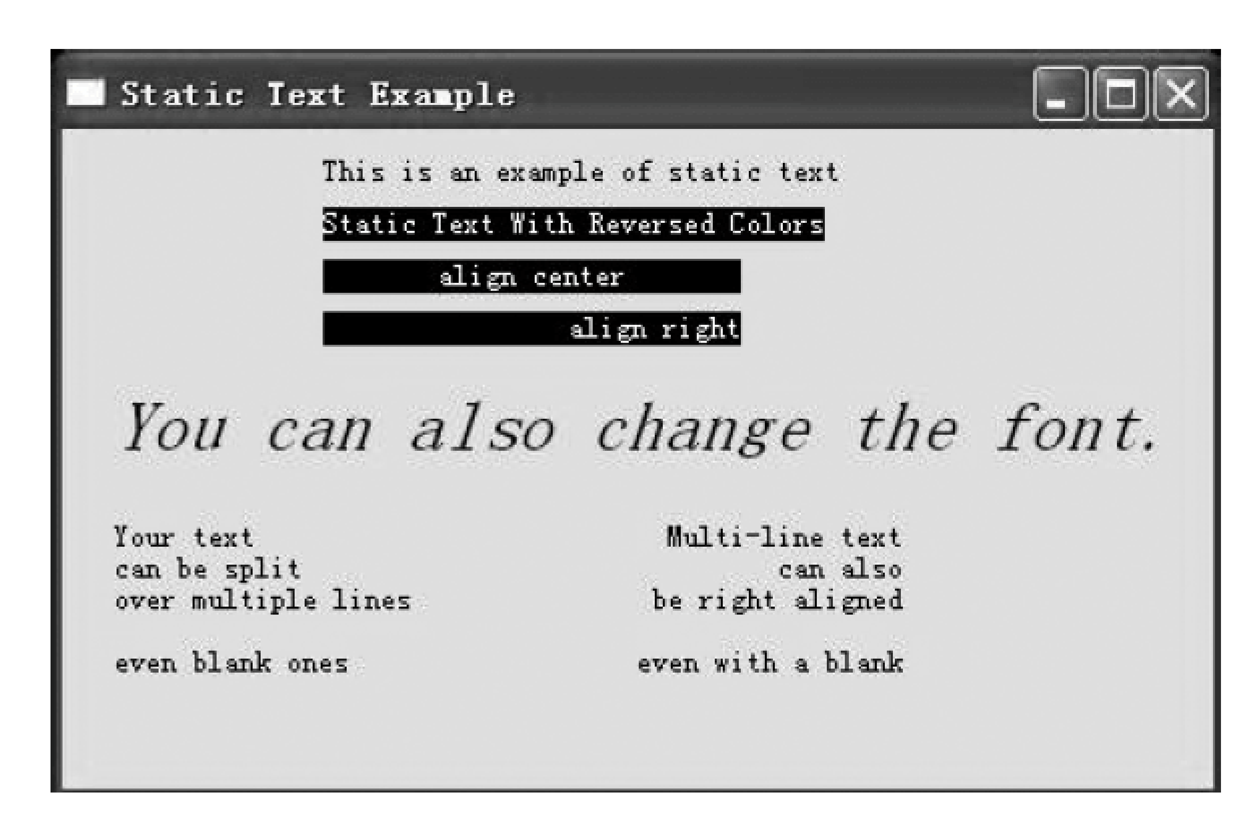


图 10.6 例 10-4 程序运行结果

10.4.2 输入文本

wxPython 使用类 wx.TextCtrl 用于用户交互的输入文本时,允许单行和多行文本输入,可作为密码输入控件等。

【例 10-5】 输入文本举例。

```
import wx
class TextFrame(wx.Frame):
    def __init__(self):
        wx.Frame.__init__(self, None, -1, 'Text Entry Example', size=(300, 100))
        panel=wx.Panel(self, -1)
        basicLabel=wx.StaticText(panel, -1, "No:")  #静态文本
```

```
        basicText=wx.TextCtrl(panel, -1, "I've entered some text!",
        size=(175, -1))                          #输入文本
        basicText.SetInsertionPoint(0)           #从左侧第一个位置输入
        pwdLabel=wx.StaticText(panel, -1, "Password:")
        pwdText=wx.TextCtrl(panel, -1, "password", size=(175, -1),style=wx.TE_
        PASSWORD)                                #输入文本的格式为密码格式
        sizer=wx.FlexGridSizer(cols=2, hgap=6, vgap=6)
        sizer.AddMany([basicLabel, basicText, pwdLabel, pwdText])
        panel.SetSizer(sizer)

if __name__=='__main__':
    app=wx.App()
    frame=TextFrame()
    frame.Show()
    app.MainLoop()
```

程序运行结果如图10.7所示。



图10.7 例10-5程序运行结果

【解析】 wxPython sizer是一个对象，用于管理容器中控件的布局。

```
sizer=wx.FlexGridSizer(cols=2, hgap=6, vgap=6)
#cols表示一行包含的列数,hgap表示两列相隔距离,vgap表示两行相隔距离
sizer.AddMany([basicLabel, basicText, pwdLabel, pwdText])
    #AddMany()方法用于加入所需设置的控件名称
panel.SetSizer(sizer)
    #设置面板上的控件
```

10.4.3 命令按钮

在wxPython中有很多不同类型的按钮，如文本按钮、位图按钮、开关按钮和通用按钮等，其中，命令按钮使用户通过敲击事件触发相应事件，采用Bind方法用于捕获在按钮的事件。

【例10-6】 命令按钮举例。

```
import wx
class MyFrame(wx.Frame):
    def __init__(self):
        wx.Frame.__init__(self, None, -1, " Example'", size=(300, 300))
        panel=wx.Panel(self, -1)
        wx.StaticText(panel, -1, "Pos:", pos=(10, 12))        #静态文本
        self.posCtrl=wx.TextCtrl(panel, -1, "", pos=(40, 10))  #输入文本
        self.btnClick=wx.Button(panel,-1,label="click",pos=(50,50),
        size=(50,30))
        self.btnEnd=wx.Button(panel,-1,label="end",pos=(100,50),size=(50,30))
```

```
        #使用 Bind 方法捕获面板事件,并将面板的事件与 OnMove 函数绑定
        panel.Bind(wx.EVT_MOTION,self.OnMove)
        #将命令按钮的单击事件与 f 函数绑定
        self.Bind(wx.EVT_BUTTON,self.f,self.btnClick)
        #单击 btnEnd 按钮会调用 OnEndMe 函数
        self.Bind(wx.EVT_BUTTON,self.OnEndMe,self.btnEnd)
    def OnMove(self, event):                                  #移动事件
        pos=event.GetPosition()
        self.posCtrl.SetValue("%s, %s" %(pos.x, pos.y))

    def f(self,event):
        self.posCtrl.SetValue("button click")

    def OnEndMe(self,event):
        self.Destroy()

if __name__=='__main__':
    app=wx.App()
    frame=MyFrame()
    frame.Show()
    app.MainLoop()
```

程序运行结果如图 10.8 和图 10.9 所示。

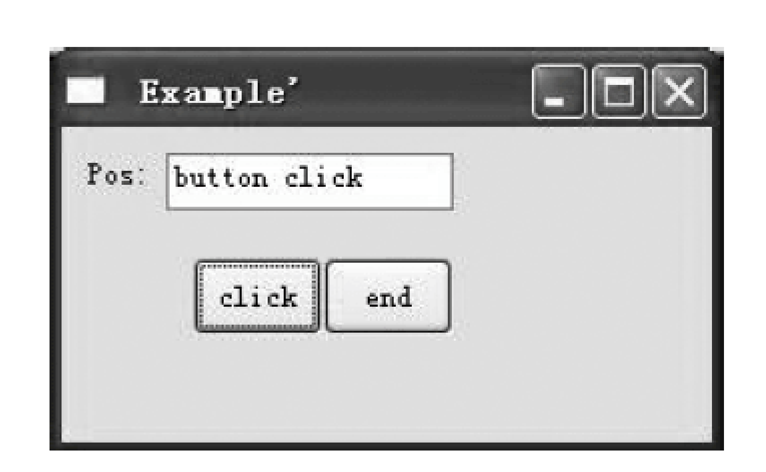


图 10.8　例 10-6 程序运行结果之一

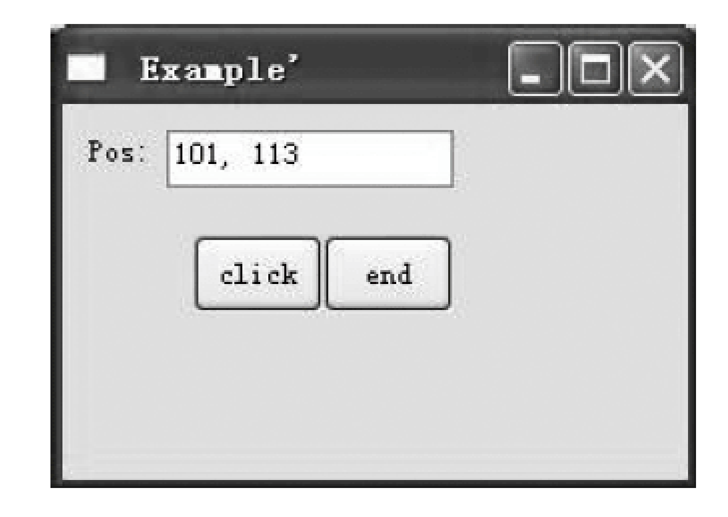


图 10.9　例 10-6 程序运行结果之二

10.4.4　滑块、微调控制框

wxPython 中滑块和微调控制框用于数字输入和显示。滑块允许用户通过在该控件的尺度内拖动指示器来选择一个数值。在 wxPython 中,该控件类是 wx.Slider。

【例 10-7】　滑块举例。

```
import wx
class SliderFrame(wx.Frame):
    def __init__(self):
        wx.Frame.__init__(self, None, -1, 'SilderFrame Example',size=(300,350))
        panel=wx.Panel(self)
```

```
        self.count=0
        self.sliderHOR=wx.Slider(panel,100,25,1,100,pos=(10,10),size=(250,-1),style=
       wx.SL_HORIZONTAL | wx.SL_AUTOTICKS | wx.SL_LABELS)
        self.sliderHOR.SetTickFreq(5, 1)
        sliderVER=wx.Slider(panel, 100, 25, 1, 100, pos=(125, 70),size=(-1, 250),style=
       wx.SL_VERTICAL | wx.SL_AUTOTICKS | wx.SL_LABELS)
        sliderVER.SetTickFreq(20, 1)
         self.posCtrl=wx.TextCtrl(panel, -1, "", pos=(10, 100))
        self.Bind(wx.EVT_SLIDER,self.f,self.sliderHOR)
    def f(self,event):
        value=self.sliderHOR.GetValue()
        self.posCtrl.SetValue("%s" %value)      #输入文本用于接收水平滑块数据
if __name__=='__main__':
    app=wx.App()
    SliderFrame().Show()
    app.MainLoop()
```

程序运行结果如图 10.10 所示。

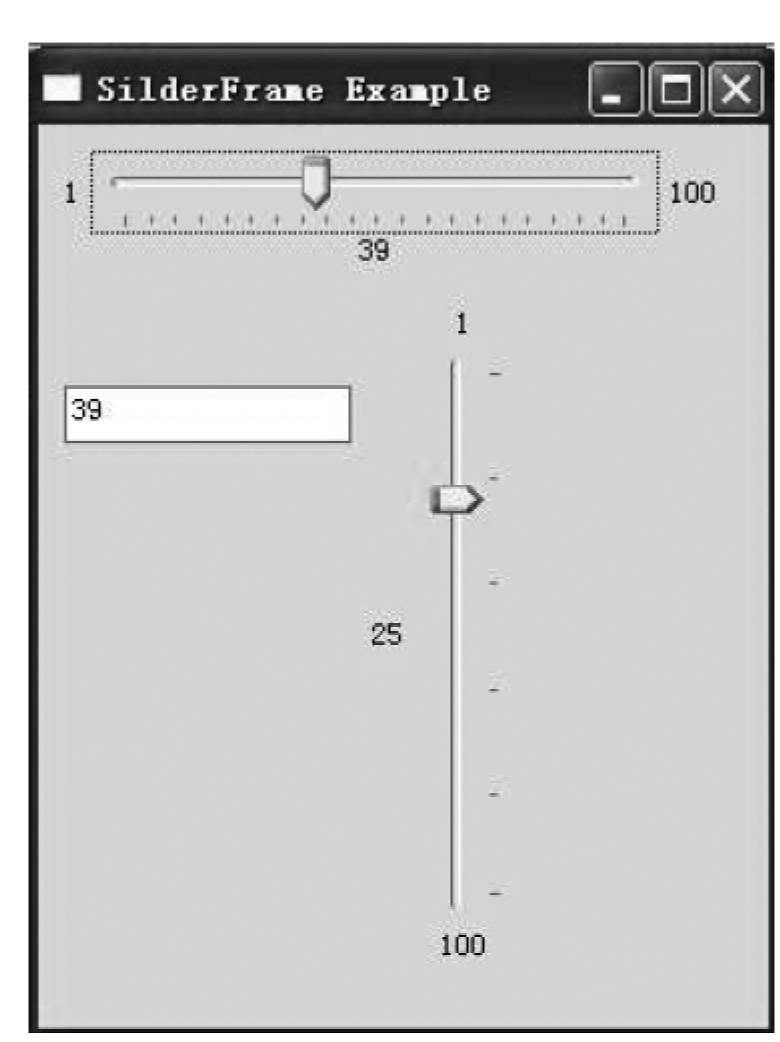


图 10.10　例 10-7 程序运行结果

【解析】 滑块提供了一个可能范围内的值的快速的、可视化的表示，但是它占据了许多的空间，另外，使用鼠标精确地设置滑块取值较为困难。因此，Python 引入了微调控制器，是文本控件和一对箭头按钮的组合，主要用于调整数字值。Python 提供了类 wx.SpinCtrl 管理微调按钮和相应的文本显示。

【例 10-8】 微调控制框举例。

```
import wx
class SpinnerFrame(wx.Frame):
    def __init__(self):
        wx.Frame.__init__(self, None, -1, 'Spinner Example',size=(100,100))
        panel=wx.Panel(self)

        self.spinner=wx.SpinCtrl(panel,-1,"",pos=(30,20),size=(50,-1))
        self.spinner.SetRange(1,100)
        self.spinner.SetValue(5)

if __name__=='__main__':
    app=wx.App()
    SpinnerFrame().Show()
    app.MainLoop()
```



图 10.11　例 10-8 程序运行结果

程序运行结果如图 10.11 所示。

10.4.5　单选钮和复选框

单选钮通常以组的形式出现，当多个单选钮构成一个组，只允许用户在其中选择一项，如选择某个人的性别等。复选框与单选钮有很多相似的地方。不过它可以以单个或一个组的形式出现，用户可以在其中选择一个，也可以选择多个。

在 wxPython 中，单选钮有两种方法可以创建。

(1) 使用 wx.RadioButton 类，一次创建一个按钮；

(2) wx.RadioBox 类可以使用单一对象来配置完整的一组按钮，这些按钮显示在一个矩形中。

本书只介绍 wx.RadioButton 类。

【例 10-9】　单选钮举例。

```
import wx
class RadioButtonFrame(wx.Frame):
    def __init__(self):
        wx.Frame.__init__(self, None, -1, 'RadioButton Example', size=(150, 200))
        panel=wx.Panel(self, -1)
        wx.RadioButton(panel, -1, "man", (35, 40), (150, 20))
        wx.RadioButton(panel, -1, "woman", (35, 60), (150, 20))

if __name__=='__main__':
    app=wx.App()
    RadioButtonFrame().Show()
    app.MainLoop()
```

程序运行结果如图 10.12 所示。

图 10.12　例 10-9 程序运行结果

【例 10-10】　复选框举例。

```
import wx
class CheckBoxFrame(wx.Frame):
    def __init__(self):
        wx.Frame.__init__(self, None, -1, 'Checkbox Example', size=(150, 200))
        panel=wx.Panel(self, -1)
        wx.CheckBox(panel, -1, "apple", (35, 40), (150, 20))
        wx.CheckBox(panel, -1, "banana", (35, 60), (150, 20))
        wx.CheckBox(panel, -1, "orange", (35, 80), (150, 20))
if __name__=='__main__':
    app=wx.App()
    CheckBoxFrame().Show()
    app.MainLoop()
```

程序运行结果如图 10.13 所示。

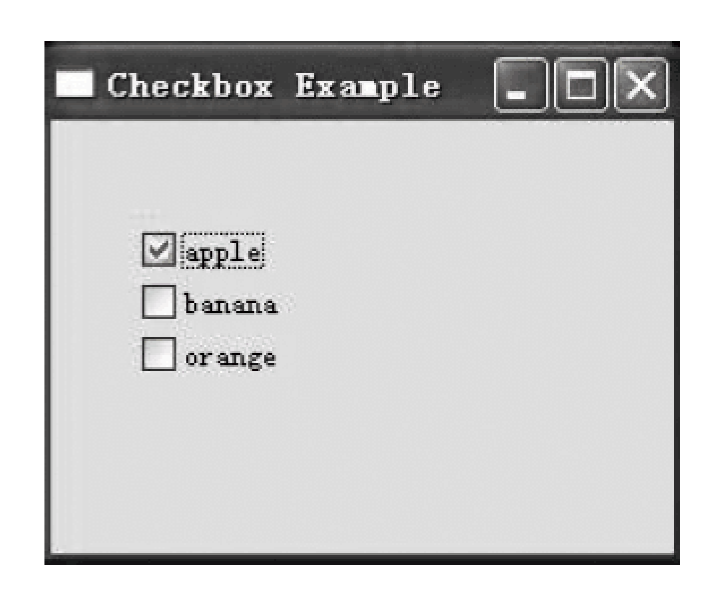


图 10.13 例 10-10 程序运行结果

10.4.6 列表框和组合框

列表框可以显示一组项目的列表,用户可以根据需要从中选择一个或多个选项。默认情况下列表框单列垂直显示所有的选项,可设置列表框为多列列表的形式,如果项目数目超过了列表框可显示的数目,控件上将自动出现滚动条。

在 wxPython 中,wx. ListBox 的构造函数如下所示:

```
wx.ListBox(parent, id, pos=wx.DefaultPosition, size=wx.DefaultSize, choices=None, style
=0, validator=wx.DefaultValidator, name="listBox")
```

列表框将显示在列表中的元素放置在参数 choices 中,它是一个字符串的序列。列表框有 3 种互斥选择类型样式:

- wx. LB_EXTENDED: 用户可以通过使用 Shift 键并单击鼠标来选择一定范围内的连续的选项,或使用等同功能的按键。
- wx. LB_MULTIPLE: 用户可以一次选择多个选项(选项可以是不连续的)。实际上,在这种情况下,列表框的行为就像是一组复选框。
- wx. LB_SINGLE: 用户一次只能选一个选项。实际上,在这种情况下,列表框的行为就像是一组单选钮。

列表框的方法如下所示:

- Append(item): 把字符串项目添加到列表框的尾部。
- Clear(): 清空列表框。
- Delete(n): 删除列表框中索引为 n 的项目。
- Deselect(n): 在多重选择列表框中,导致位于位置 n 的选项取消选中。
- FindString(string): 返回给定字符串的整数位置,如果没有发现则返回-1。
- GetCount(): 返回列表中字符串的数量。
- GetSelection()得到当前选择项的整数索引(仅对于单选列表)。
- GetSelections()返回多选列表所选项目的整数位置的元组。
- GetStringSelection()返回单选列表当前选择的字符串。
- GetString(n) : 得到位置 n 处的字符串。
- SetString(n, string): 设置位置 n 处的字符串。

- InsertItems(items, pos)：插入参数 items 中的字符串列表到该列表框中 pos 参数所指定的位置前。位置 0 表示把项目放在列表的开头。
- Selected(n)：返回对应于索引为 n 的项目的选择状态的布尔值。
- Set(choices)：重新使用 choices 的内容设置列表框。

【例 10-11】 列表框举例。

```
import wx
class ListBoxFrame(wx.Frame):
    def __init__(self):
        wx.Frame.__init__(self, None, -1, 'ListBoxFrame Example', size=(150, 200))
        panel=wx.Panel(self, -1)
        sampleList=['zero', 'one', 'two', 'three', 'four', 'five','six', 'seven', 'eight
', 'nine', 'ten']
        wx.StaticText(panel, -1, "EXTENDED ", (40, 10))
        listBox1=wx.ListBox(panel, -1, (40, 40), (80, 120), sampleList, wx.LB_EXTENDED)
        wx.StaticText(panel, -1, "MULTIPLE ", (130, 10))
        listBox2=wx.ListBox(panel, -1, (130, 40), (80, 120), sampleList, wx.LB_MULTIPLE)
        wx.StaticText(panel, -1, "SINGLE ", (220, 10))
        listBox3=wx.ListBox(panel, -1, (220, 40), (80, 120), sampleList, wx.LB_SINGLE)

if __name__=='__main__':
    app=wx.App()
    ListBoxFrame().Show()
    app.MainLoop()
```

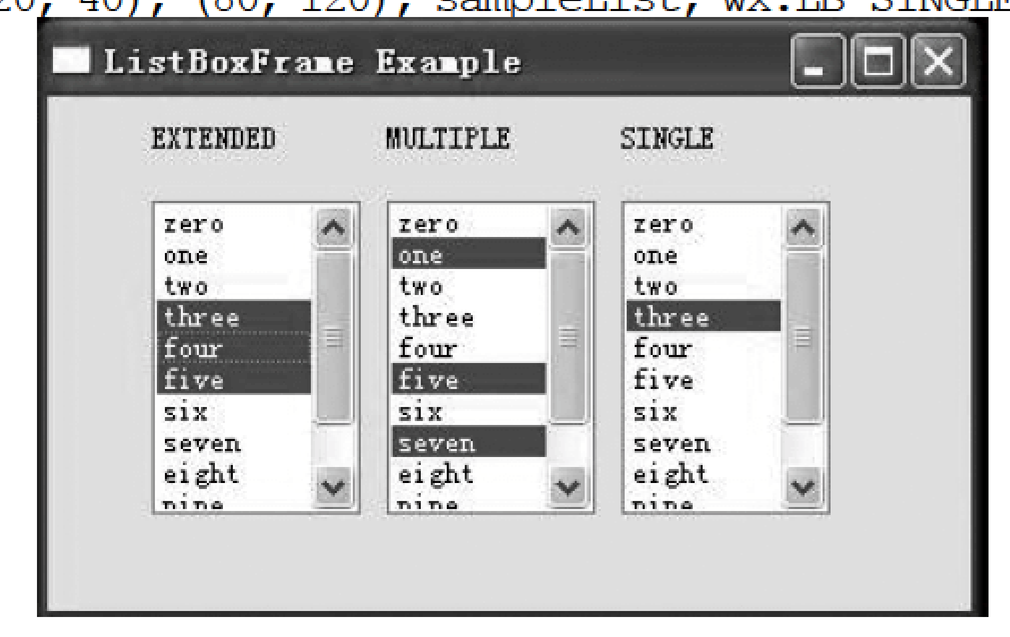


图 10.14　例 10-11 程序运行结果

程序运行结果如图 10.14 所示。

【例 10-12】 组合框举例。



```
import wx
class ComboxFrame(wx.Frame):
    def __init__(self):
        wx.Frame.__init__(self, None, -1, 'Combox Example',size=(350,300))
        panel=wx.Panel(self)
        sampleList=['zero', 'one', 'two', 'three', 'four', 'five','six',
        'seven', 'eight', 'nine', 'ten']
        wx.StaticText(panel, -1, "Select one:", (15, 15))
        wx.ComboBox(panel, -1, "default value", (15, 30), wx.DefaultSize,
        sampleList, wx.CB_DROPDOWN)
        wx.ComboBox(panel, -1, "default value", (150, 30), wx.DefaultSize,
        sampleList, wx.CB_SIMPLE)

if __name__=='__main__':
    app=wx.App()
    ComboxFrame().Show()
```

```
    app.MainLoop()
```

程序运行结果如图 10.15 所示。

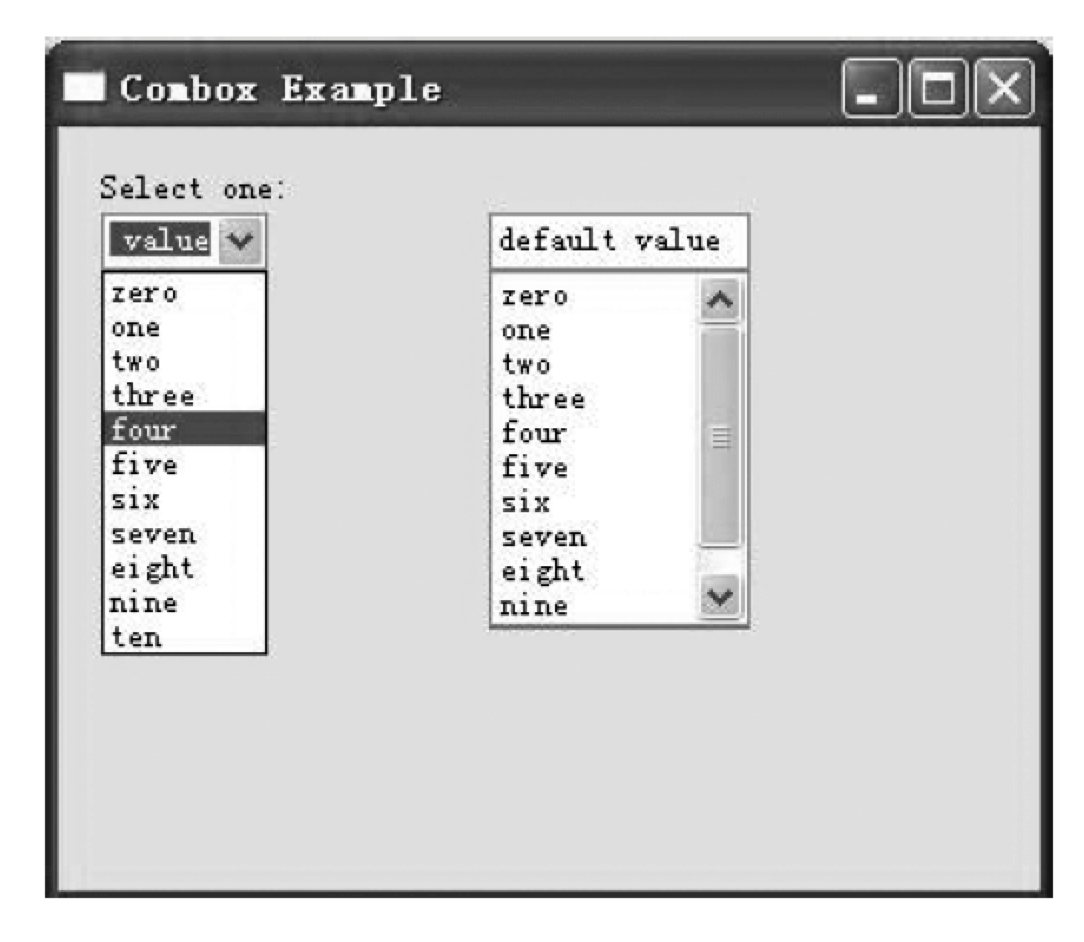


图 10.15　例 10-12 程序运行结果

10.4.7　菜单

作为用户界面设计重要的组成部分，菜单起到主控模块的作用，负责调用各功能模块，实现软件系统的功能划分，如图 10.16 所示。

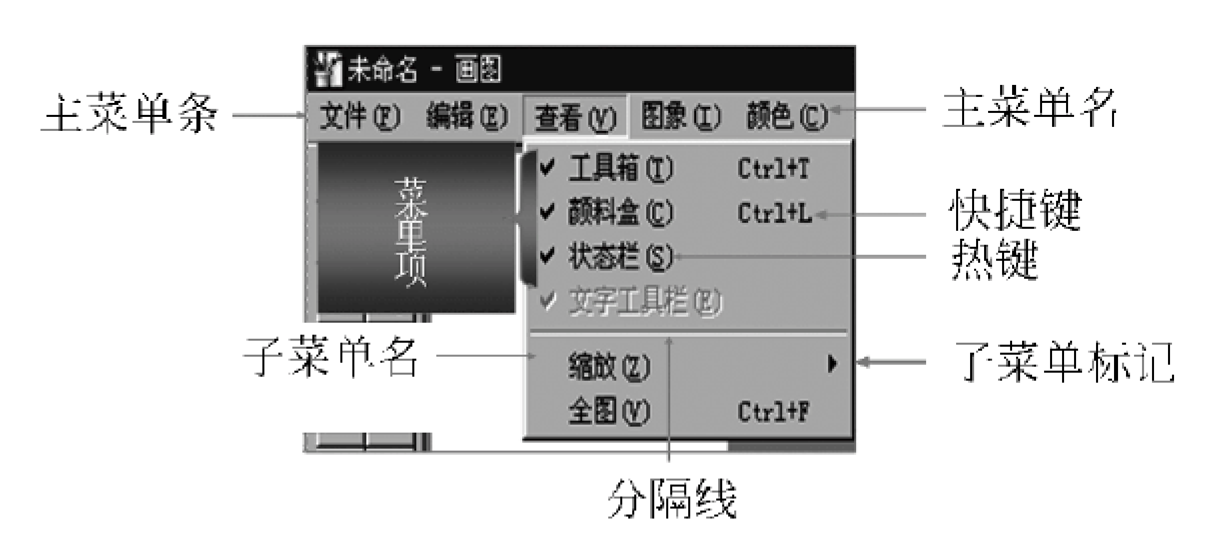


图 10.16　菜单

在 wxPython 中有 3 个主要的类实现菜单功能。类 wx.MenuBar 管理菜单栏自身；wx.Menu 管理一个单独的下拉或弹出菜单，一个 wx.MenuBar 实例可以包含多个 wx.Menu 实例；类 wx.MenuItem 代表一个 wx.Menu 中的一个特定项目。

Python 创建菜单操作过程如下：

(1) 创建菜单栏，将菜单栏附加给框架；

(2) 创建单个的菜单，将菜单附加给菜单栏或一个父菜单；

(3) 创建单个的菜单项，把这些菜单项附加给适当的菜单，为每个菜单项创建一个事件绑定。

菜单 wx.MenuBar 方法如表 10.2 所示。

表 10.2　菜单 wx.MenuBar 方法

Append(menu,title)	将 menu 参数添加到菜单栏的尾部(靠右显示)。title 参数显示新的菜单。如果成功返回 True,否则返回 False
Insert(pos,menu,title)	将给定的 menu 插入到指定的位置 pos(调用这个函数之后,GetMenu(pos)==menu 成立)。就像在列表中插入一样,所有后面的菜单被右移。菜单的索引以 0 开始,所以 pos 为 0 等同于将菜单放置于菜单栏的左端。使用 GetMenuCount()作为 pos 等同于使用 Append。title 显示名字。函数如果成功则返回 True
Remove(pos)	删除位于 pos 的菜单,之后的其他菜单左移。函数返回被删除的菜单
Replace(pos,menu,title)	使用给定的 menu 和 title 替换位置 pos 处的菜单。菜单栏上的其他菜单不受影响。函数返回替换前的菜单

【例 10-13】 菜单举例。

```
import wx
class MenuFrame(wx.Frame):
    def __init__(self):
        wx.Frame.__init__(self, None, -1, 'Menu Example')
        panel=wx.Panel(self)

        menuBar=wx.MenuBar()                #菜单栏
        menu=wx.Menu()                      #菜单
        menu.Append(10,"&New")
        menu.Append(12,"&Open")
        menu.Append(13,"&Save")
        menu.AppendSeparator()
        self.exit=menu.Append(-1,"&Exit")
        self.Bind(wx.EVT_MENU, self.OnExit,self.exit)
        menuBar.Append(menu, "&File")       #将菜单 menu 添加到菜单栏,显示为 File
        menu2=wx.Menu()
        menu2.Append(21,"&Copy")
        menu2.Append(22,"&Paste")
        menu2.Append(23,"&Cut")
        menuBar.Append(menu2, "&Edit")      #将菜单 menu2 添加到菜单栏,显示为 Edit
        self.SetMenuBar(menuBar)            #将菜单栏添加到框架上

    def OnExit(self, event):
        self.Close(True)

if __name__=='__main__':
    app=wx.App()
    MenuFrame().Show()
    app.MainLoop()
```

程序运行结果如图 10.17 所示。

菜单是用户访问应用程序功能的主要入口,设计菜单一般应遵守如下准则:

(1) 使菜单有均衡的长度

菜单的长度应该基本一致,其所包含的项目的最大数量一般在10～15。

(2) 创建合理的项目组

分隔符之间的菜单项为一组,其所包含的菜单项不应多于5项。

(3) 菜单的顺序要遵循标准

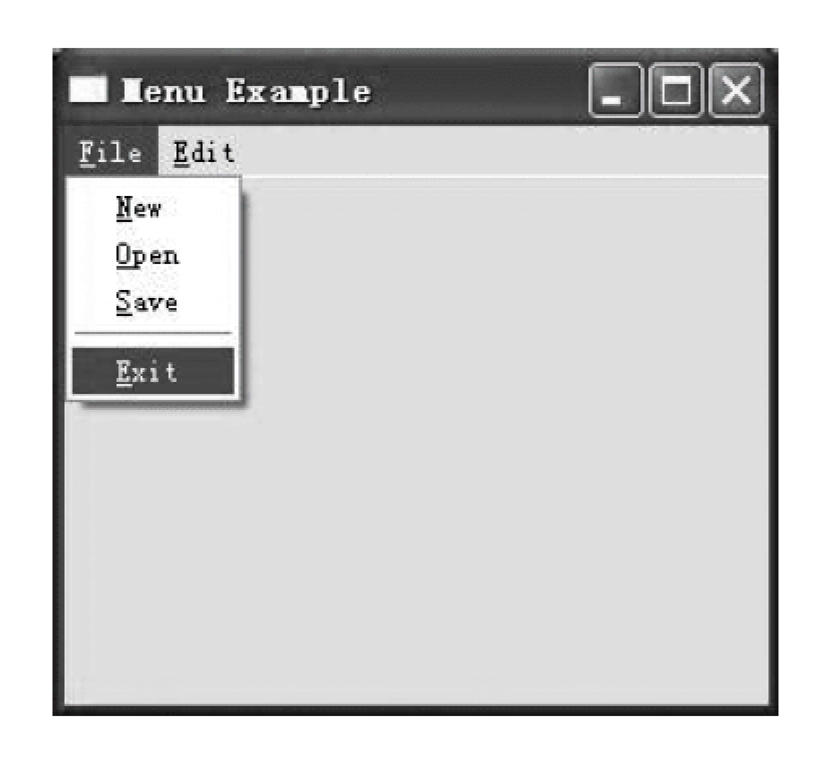


图 10.17　例 10-13 程序运行结果

菜单的顺序应该遵循公认的标准。最左边的菜单应该是 File(文件),包含 New(新建)、Open(打开)、Save(保存)、Print(打印)和 Quit(退出)功能。下一个菜单是 Edit(编辑),包含 Undo(撤销)、Cut(剪切)、Copy(复制)、Paste(粘贴)等。

(4) 为常使用的项目提供方便的访问

更常用的菜单选项应放在顶部,便于用户使用。

(5) 使用有含义的菜单名称

① 菜单打开的宽度与所包含菜单项目的最长的名字成正比。

② 尽量避免使顶级菜单的名字少于4个字母。

(6) 调用对话框的菜单项应带有省略号

任何会导致一个对话框被显示的菜单项,都应该有一个以省略号(...)结尾的标签。

(7) 使用标准的快捷键

对快捷键,使用通常功能的公认的标准,如表10.3所示。

表 10.3　快捷键功能

Ctrl+a	全选	Ctrl+n	新建	Ctrl+s	保存
Ctrl+c	复制	Ctrl+o	打开	Ctrl+v	粘贴
Ctrl+f	查找	Ctrl+p	打印	Ctrl+w	关闭
Ctrl+g	查找下一个	Ctrl+q	退出	Ctrl+x	剪切

10.4.8　工具栏和状态栏

随着软件的功能越来越多,用户界面变得越来越复杂,特别是菜单的嵌套层次越来越深,用户使用起来很不方便。为此将一些常用控件作为子项放在工具栏中,工具栏一般位于菜单栏的下方,通过各个子项与菜单中的项对应,给用户提供应用程序常用菜单的快捷访问。

【例 10-14】　工具栏举例。

```
import wx
import wx.py.images
```

```
class ToolBarFrame(wx.Frame):
    def __init__(self):
        wx.Frame.__init__(self, None, -1, 'ToolBar Example')
        panel=wx.Panel(self)
        panel.SetBackgroundColour('red')

        statusBar=self.CreateStatusBar()            #创建状态栏

        toolbar=self.CreateToolBar()                #创建工具栏
        self.toolQuit=toolbar.AddSimpleTool(wx.NewId(), wx.py.images.getPyBitmap(), "
        quit", "Quit,,,,,")
        toolbar.Realize()
        self.Bind(wx.EVT_TOOL, self.OnQuit,self.toolQuit)
    def OnQuit(self,event):
        self.Close()
if __name__=='__main__':
    app=wx.App()
    ToolBarFrame().Show()
    app.MainLoop()
```

程序运行结果如图 10.18 所示。

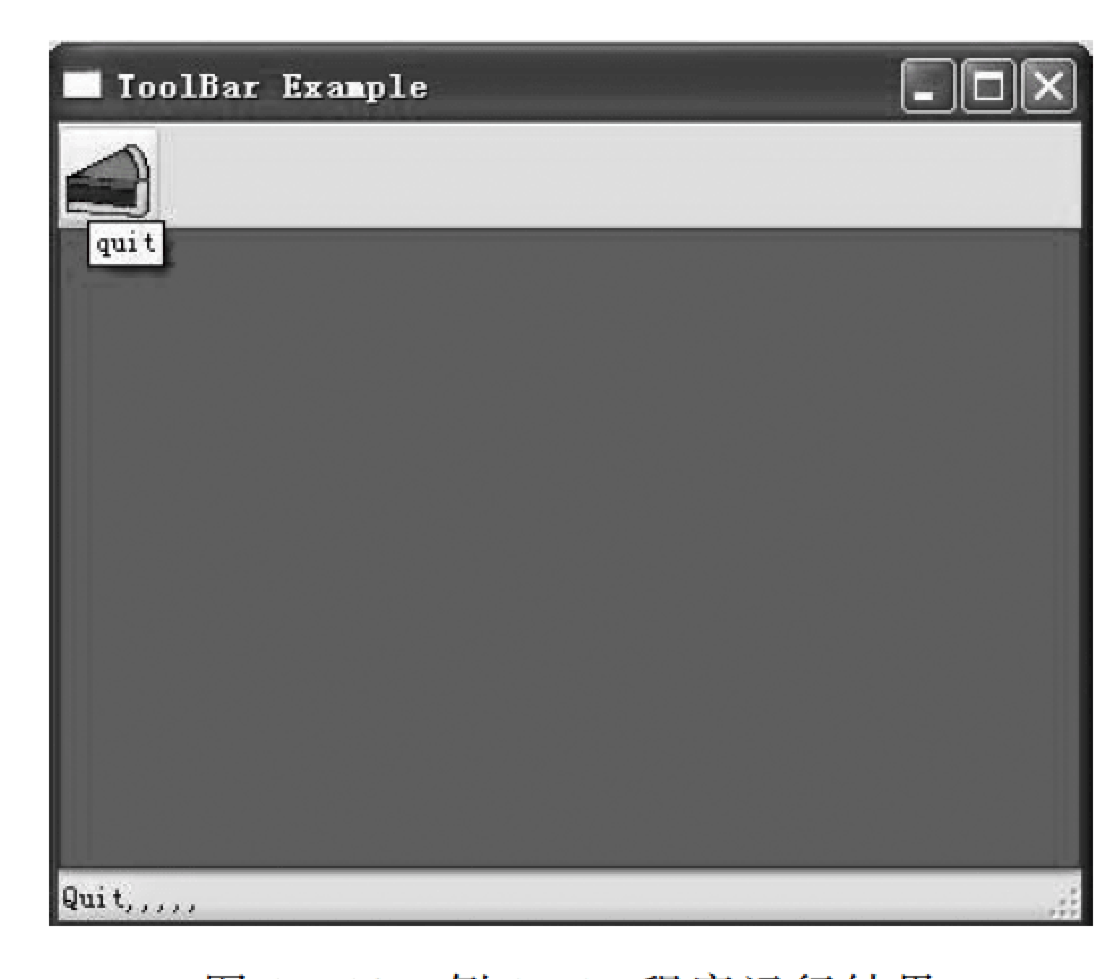


图 10.18 例 10-14 程序运行结果

10.5 对话框

对话框用于应用程序与用户之间进行信息交互，当对话框中的信息必须输入时，将阻塞别的控件接收用户事件，直到该对话框关闭。wxPython 提供了警告对话框(wx.MessageDialog)、单行文本对话框(wx.TextEntryDialog)和列表选择对话框(wx.SingleChoiceDialog)等对话框。

10.5.1 警告对话框

Python 提供的警告对话框与 Visual Basic 语言提供的 MessageBox 函数功能类似，用于告知用户警告信息。通过 wx.MessageDialog 设置对话框的消息和按钮，构造函数如下：

```
wx.MessageDialog(parent, message, caption="Message box",
style=wx.OK | wx.CANCEL, pos=wx.DefaultPosition)
```

wx.MessageDialog 的按钮样式如表 10.4 所示。

表 10.4 wx.MessageDialog 的按钮样式

wx.CANCEL	包括一个 cancel(取消)按钮。这个按钮有一个 ID 值 wx.ID_CANCEL
wx.NO_DEFAULT	在一个 wx.YES_NO 对话框中，No(否)按钮是默认的
wx.OK	包括一个 OK 按钮，这个按钮有一个 ID 值 wx.ID_OK
wx.YES_DEFAULT	在一个 wx.YES_NO 对话框中，Yes 按钮是默认的，这是默认行为
wx.YES_NO	包括 Yes 和 No 按钮，各自的 ID 值分别是 wx.ID_YES 和 wx.ID_NO

【例 10-15】 警告对话框举例。

```
import wx
class MessageBoxFrame(wx.Frame):
    def __init__(self):
        wx.Frame.__init__(self, None, -1, 'MessageBox Example',
        size=(400,275))
        panel=wx.Panel(self, -1)

        self.btnExit=wx.Button(panel,-1,label="Exit",pos=(330,200),
        size=(50,30))
        self.Bind(wx.EVT_BUTTON,self.OnExit,self.btnExit)
    def OnExit(self,event):
        dialog=wx.MessageDialog(self,"Exit?","MessageDialog",
        wx.OK|wx.CANCEL|wx.ICON_QUESTION)
        result=dialog.ShowModal()       #以模式框架的方式显示
        if result==wx.ID_OK:
            dialog.Destroy              #销毁对话框
            self.Destroy
if __name__=='__main__':
    app=wx.App()
    MessageBoxFrame().Show()
    app.MainLoop()
```

程序运行结果如图 10.19 所示。



图 10.19　例 10-15 程序运行结果

10.5.2　单行文本对话框

单行文本对话框从用户得到短的文本输入，通常用于用户名或密码的输入，Python 提供 wx.TextEntryDialog 实现。

【例 10-16】　单行文本对话框举例。

```
import wx
class TextEntryDialogFrame(wx.Frame):
    def __init__(self):
        wx.Frame.__init__(self, None, -1, 'TextEntryDialog Example',size=(400,275))
        panel=wx.Panel(self, -1)

        dialog=wx.TextEntryDialog(self,"input?","TextEntryDialog", "zhou")
        result=dialog.ShowModal()
        if result==wx.ID_OK:
            dialog.Destroy
            self.Destroy
if __name__=='__main__':
    app=wx.App()
    TextEntryDialogFrame().Show()
    app.MainLoop()
```

程序运行结果如图 10.20 所示。

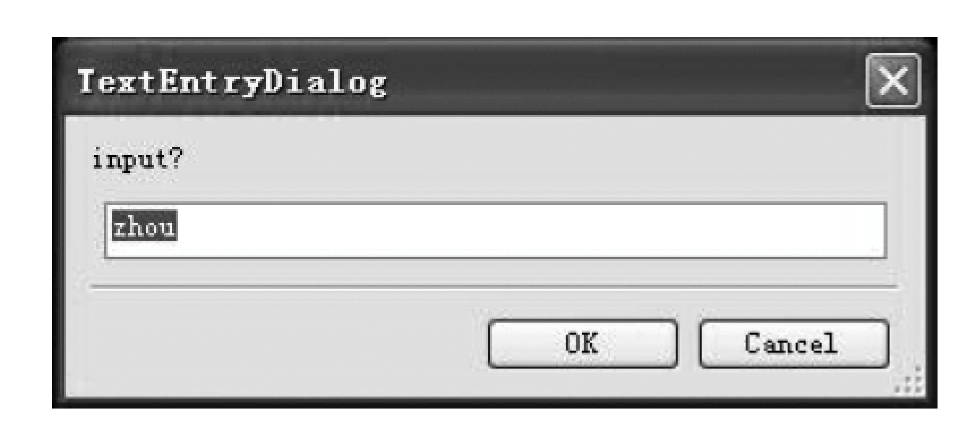


图 10.20　例 10-16 程序运行结果

10.5.3　列表选择对话框

列表选择对话框使用户能够从一个有效选项列表中进行选择，Python 提供 wx.SingleChoiceDialog 实现。

【例 10-17】　列表选择对话框举例。

```
import wx
```

```
class SingleChoiceDialogFrame(wx.Frame):
    def __init__(self):
        wx.Frame.__init__(self, None, -1, 'SingleChoiceDialog Example',
        size=(400,275))
        panel=wx.Panel(self, -1)

        sampleList=['2.0', '2.1.3', '2.2', '2.3.1', '2.7.3', '3.3.0']

        dialog=wx.SingleChoiceDialog(self,"Python version is ?",
        "SingleChoiceDialog", sampleList)
        result=dialog.ShowModal()
        if result==wx.ID_OK:
            response=dialog.GetStringSelection()
            dialog.Destroy
            self.Destroy
if __name__=='__main__':
    app=wx.App()
    SingleChoiceDialogFrame().Show()
    app.MainLoop()
```

程序运行结果如图 10.21 所示。



图 10.21　例 10-17 程序运行结果

10.6　习题

1. 建立应用程序，窗体如图 10.22 所示，有一个简单组合框、4 个命令按钮、一个文本框和一个标签。要求：

（1）单击“添加”按钮可将输入的内容添加到组合框中。

（2）单击“删除”按钮可删除组合框中选定的项目。

（3）单击“统计人数”按钮，可将组合框中的项目总数输出到右边的文本框。

（4）单击“退出”按钮或按 Esc 键退出程序。



图 10.22　习题 10.6.1 程序运行界面

2. 程序运行界面如图 10.23 所示，在窗体左边

的列表框中生成 10 个由小到大排列的 10～100 之间的随机整数。选择“右移”命令，则左边列表框的 10 个数移动到右边的列表框中；反之，将右边的列表框中的整数通过“左移”命令移至左边的列表框中，采用弹出式菜单实现。

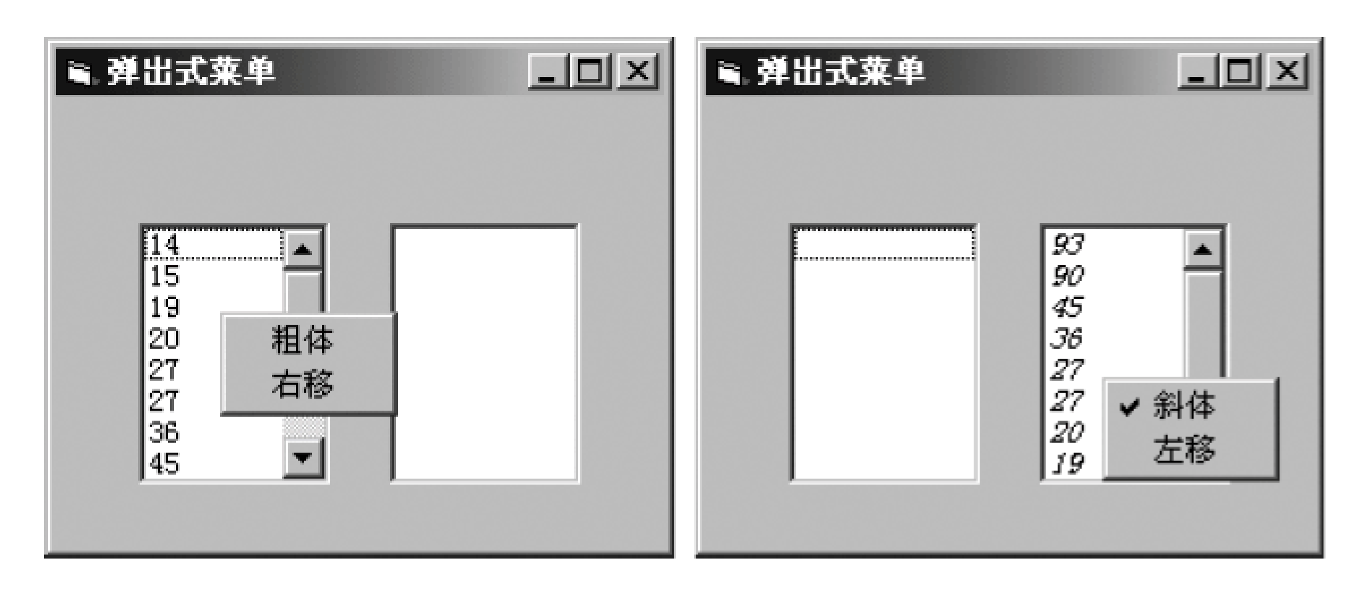
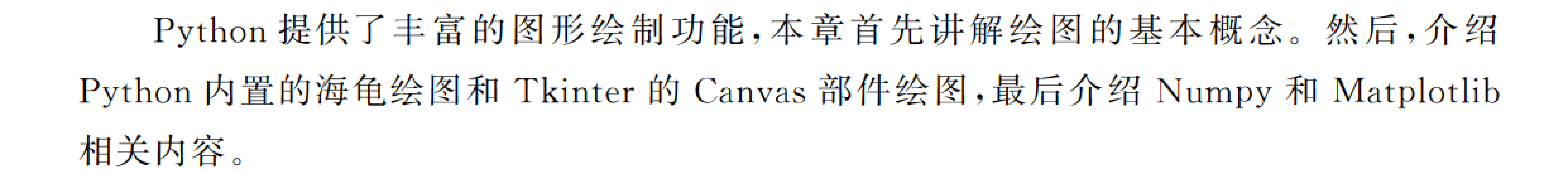


图 10.23　习题 10.6.2 程序运行界面

第11章 绘　　图



Python 提供了丰富的图形绘制功能，本章首先讲解绘图的基本概念。然后，介绍 Python 内置的海龟绘图和 Tkinter 的 Canvas 部件绘图，最后介绍 Numpy 和 Matplotlib 相关内容。

11.1　绘图概念

11.1.1　绘图简介

Python 绘图方式很多，有内置海龟绘图（turtle）模块、内置 Tkinter 模块中的画布控件（Canvas），另外还有许多开源模块，如 Matplotlib，Chaco，Python Google Chart，PyCha，pyOFC2，PyChart，PLplot，ReportLab 和 VPython 等。

11.1.2　坐标系

一个坐标系统包括坐标原点、坐标度量单位、坐标轴的长度与方向。Python 坐标系的原点在屏幕左上角，横向向右为 x 的正向，纵向向下为 y 轴的正向，如图 11.1 所示。笛卡儿直角坐标系的原点在屏幕中心点，y 轴的正向向上，如图 11.2 所示。一个点或者一条线在不同的坐标系绘制，会产生不同的效果。因此，Python 绘图应首先确定坐标系。

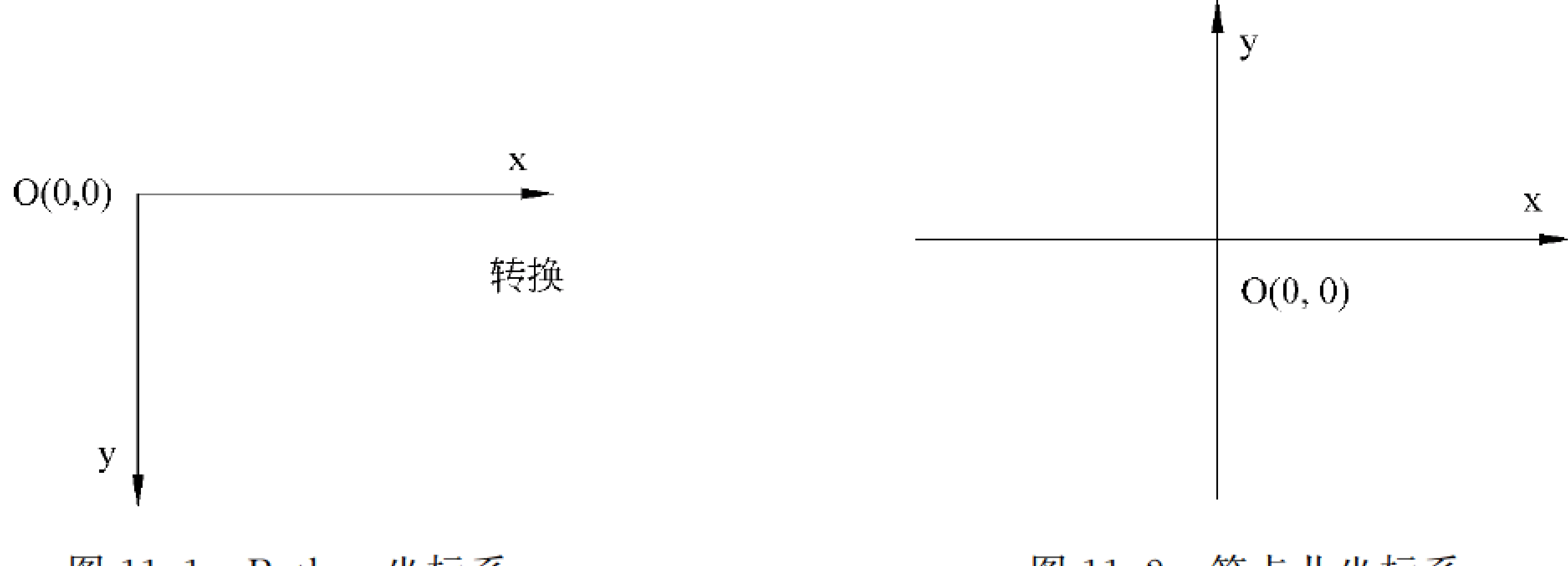


图 11.1　Python 坐标系　　　图 11.2　笛卡儿坐标系

11.2 海龟绘图

11.2.1 turtle 绘图方法

Python2.6 版本之后，内置海龟绘图工具(turtle Graphics)。turtle 的坐标轴是笛卡儿坐标系，即以中心点为原点，坐标轴与 x，y 轴类似，右为 x 轴正轴，上为 y 轴正轴。

海龟作为绘图的画笔，具有如下属性和行为。

(1) 海龟属性

海龟具有颜色、宽度等属性。

```
color(colorstring)              #绘制图形的颜色
fillcolor(colorstring)          #绘制图形的填充颜色
pensize(width)                  #绘制图形的宽度
```

(2) 海龟行为

海龟绘图命令大致分为运动命令和画笔控制命令两种。

① 运动命令

```
forward(len)                    #向前移动距离 len 代表距离
backward(len)                   #向后移动距离 len 代表距离
right(degree)                   #向右旋转 degree 度
left(degree)                    #向左旋转 degree 度
goto(x,y)                       #将画笔移动到坐标为 x,y 的位置
speed(speed)                    #画笔绘制的速度范围[0,11]整数
reset()                         #将海龟回到坐标原点
```

② 画笔控制命令

```
down()                          #代表放下画笔,开始绘图
up()                            #提起画笔,暂时不画画
```

turtle 画图步骤如下所示：

步骤一：引入 turtle。

```
from turtle import *            #将 turtle 中的所有方法导入
```

步骤二：绘制各种图形，如线条、多边形、弧、圆等。

步骤三：结束绘制。

```
s=Screen()
s.exitonclick()
```

11.2.2 实例讲解

【例 11-1】 绘制水平线。

```
from turtle import *
def jumpto(x,y):
    up()
    goto(x,y)
    down()
reset()
colorlist=['red','green','yellow','purple']
for i in range(4):
    jumpto(-110,50-i*50)
    width(5*(i+1))
    color(colorlist[i])
    forward(200)
s=Screen()
s.exitonclick()
```

程序运行结果如图 11.3 所示。

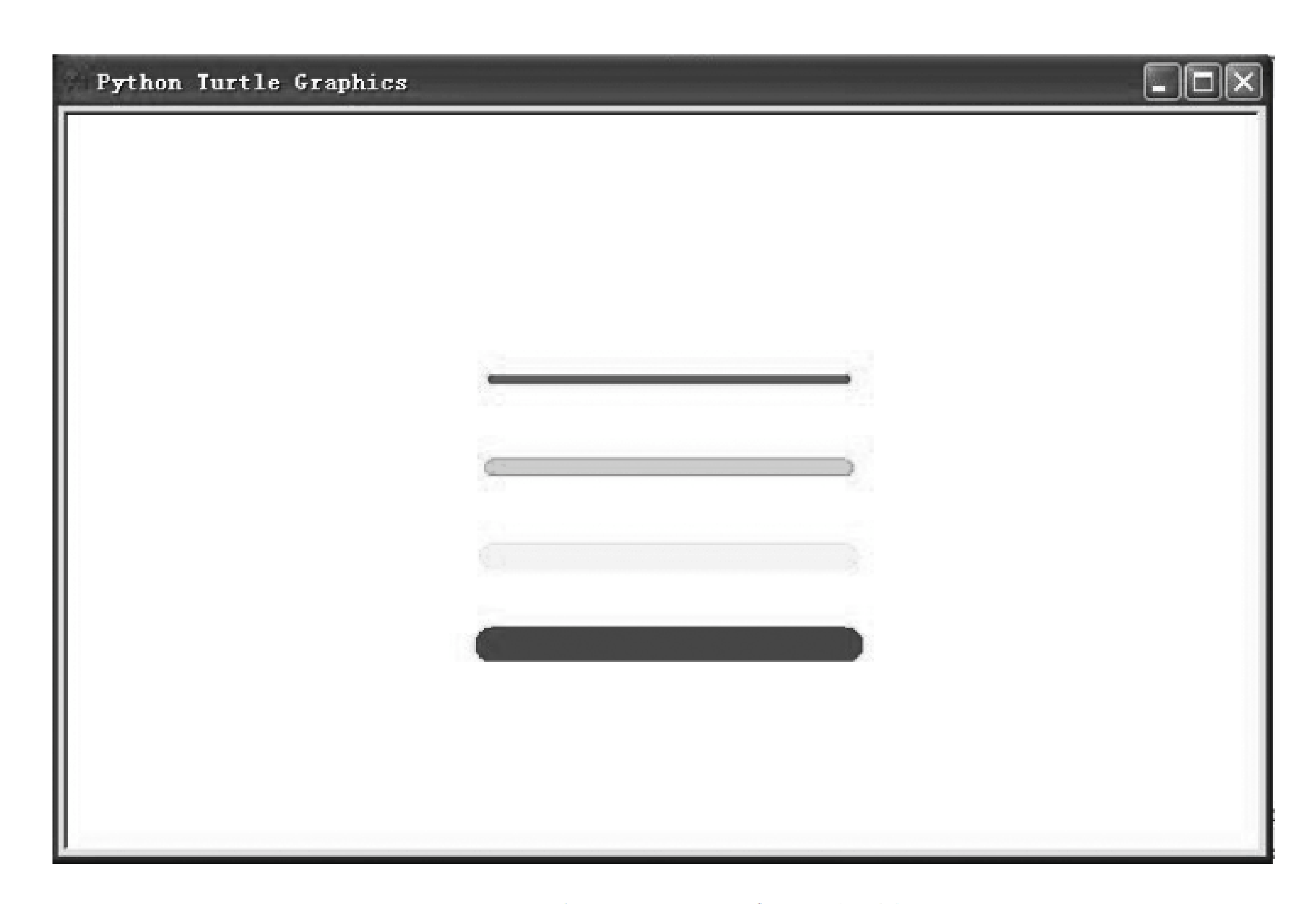


图 11.3　例 11-1 程序运行结果

【例 11-2】 绘制正方形。

```
from turtle import *
reset()
for i in range(4):
    width(11)
    color("purple")
    forward(110)          #向前移动距离
    right(90)             #向右旋转 90 度
up();goto(50,-150); down()
write("Square")
s=Screen()
s.exitonclick()
```

程序运行结果如图 11.4 所示。

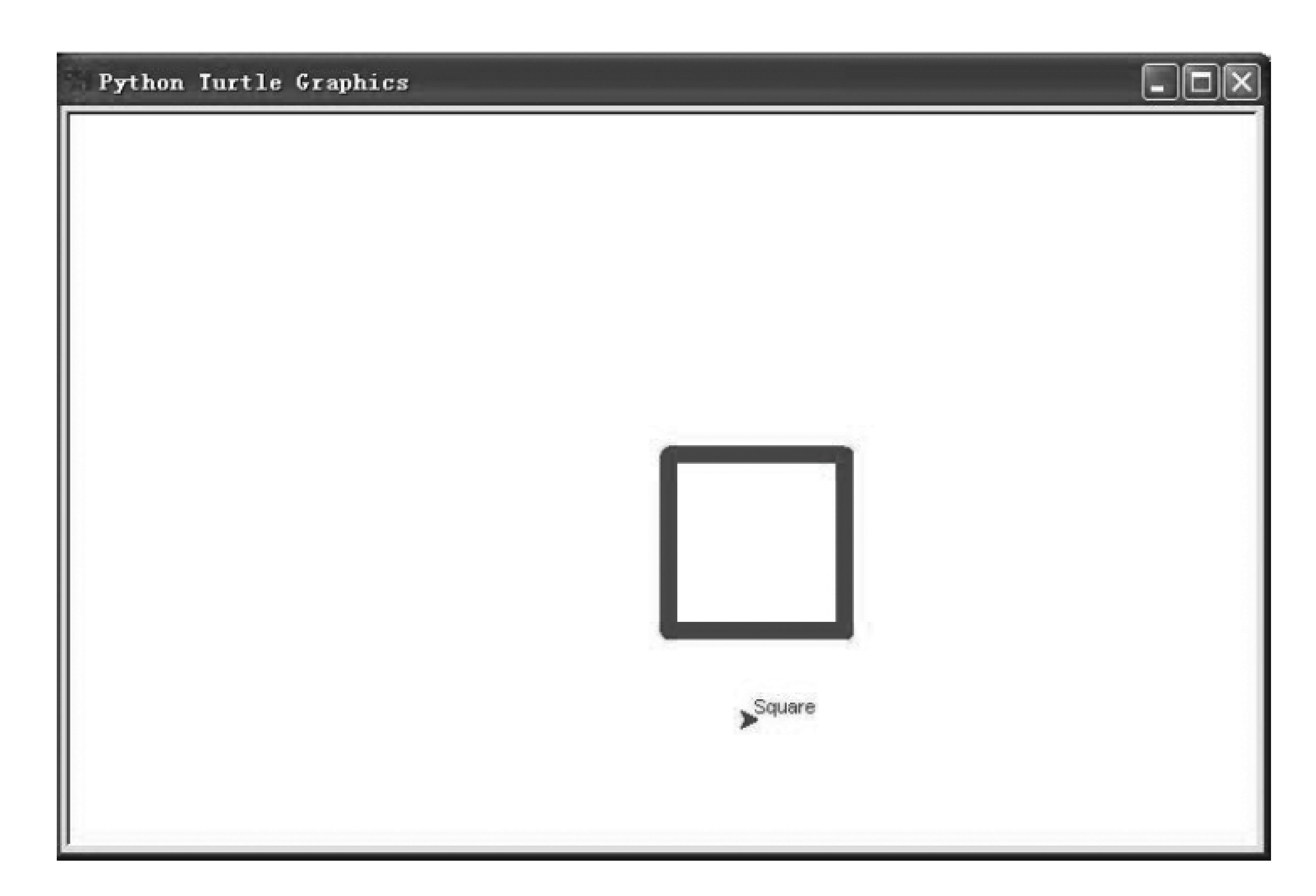


图 11.4　例 11-2 程序运行结果

【例 11-3】 绘制五角星。

```
from turtle import *
reset()
color("purple")
pensize(3)
goto(0,0)
speed(3)
for i in range(6):
    forward(110)
    right(144)
color("red")
up();goto(50,-120);down()
write("circle")
s=Screen()
s.exitonclick()
```

程序运行结果如图 11.5 所示。



图 11.5　例 11-3 程序运行结果

【例 11-4】 绘制圆。

turtle 内置画圆函数 circle(r),在指定位置,绘制半径为 r 圆。

```
from turtle import *
circle(50)
up();goto(0,100);down()
circle(50)
s=Screen()
s.exitonclick();
```

程序运行结果如图 11.6 所示。

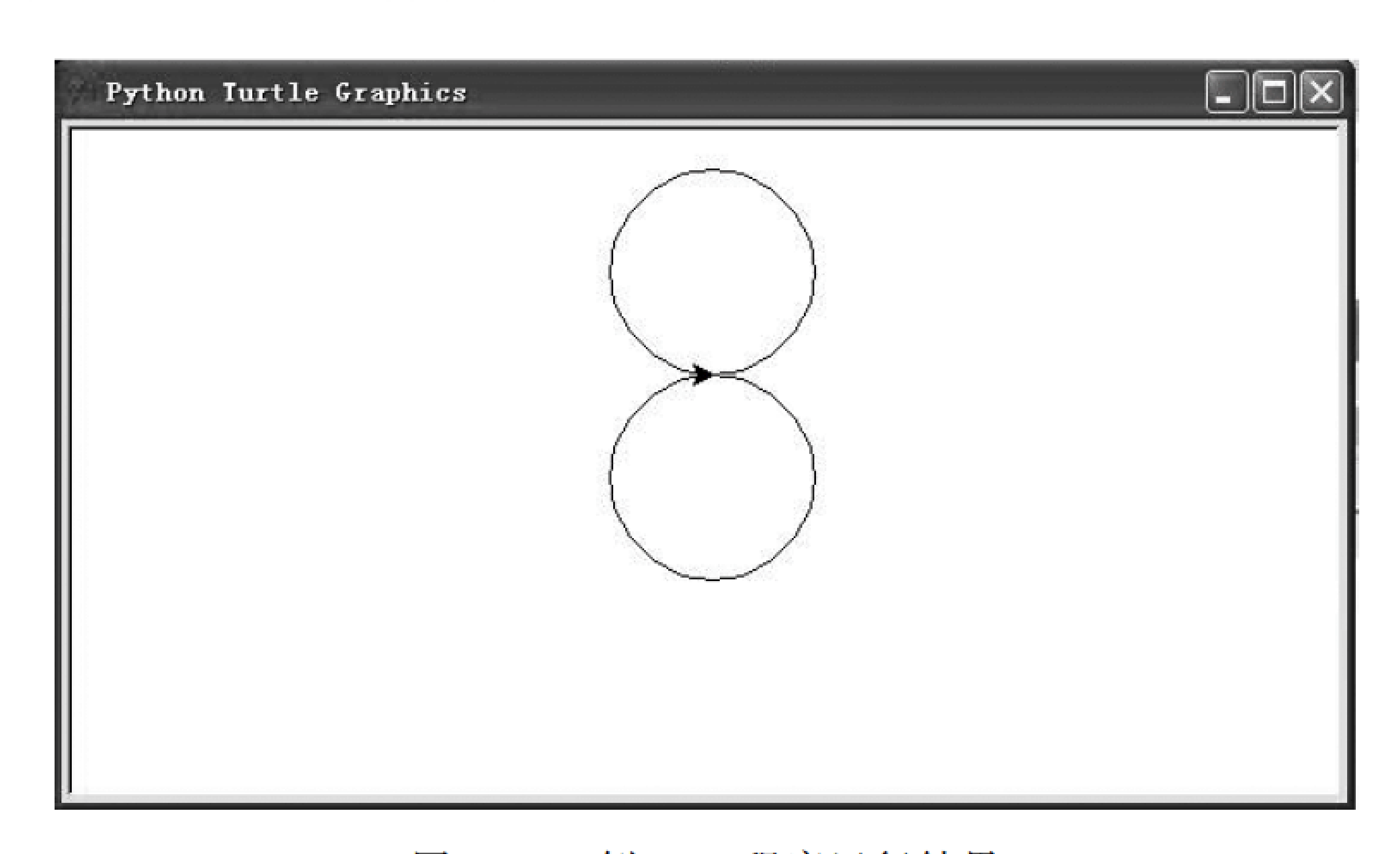


图 11.6　例 11-4 程序运行结果

11.3　Canvas 绘图

11.3.1　Canvas 绘图方法

Tkinter 作为 Python 的内置标准 GUI 库,其部件 Canvas 是画布控件,用于显示图形元素等,Canvas 坐标系为 Python 坐标系。

Canvas 画图步骤如下所示:

步骤一:引入 Tkinter。

```
import Tkinter
```

步骤二:设置画布大小,背景色等属性。

```
top=Tkinter.Tk()
c=Tkinter.Canvas(top, bg, height, width)
```

参数说明:bg 为背景色,如 bg="red"等。

步骤三:绘画各种图形,如线条、多边形、弧、圆等。

(1) 创建一条线条目。

```
line=canvas.create_line(x0, y0, x1, y1, ..., xn, yn, options)
```

参数说明：x0,y0 为第 1 个点的位置坐标；x1,y1 为第 2 个点的位置坐标，依此类推。options 为可选项，是填充色，如 fill="red"等。

(2) 创建一个多边形，必须有至少 3 个顶点。

```
polygon=canvas.create_polygon(x0, y0, x1, y1,…,xn, yn, options)
```

(3) 创建弧，可以是一个和弦、饼图扇区，或是一个简单的弧。

```
arc=canvas.create_arc(x0, y0, x1, y1, start, extent, options)
```

参数说明：x0,y0 为第 1 个点的位置坐标；x1,y1 为第 2 个点的位置坐标。

- Start 为开始旋转角度。
- Extent 为旋转角度。
- options 为可选项，是填充色，如 fill="red"等。

(4) 创建一个圆或椭圆。

```
oval=canvas.create_oval(x0, y0, x1, y1, options)
```

步骤四：消息主循环。

```
c.pack()
top.mainloop()
```

11.3.2 实例讲解

【例 11-5】 Canvas 举例。

```
import Tkinter
top=Tkinter.Tk()
c=Tkinter.Canvas(top, bg="blue", height=500, width=500)
line=c.create_line(0, 0, 110, 110, fill="red")
#绘制一条直线,从原点到(110,110),填充色为红色
polygon=c.create_polygon(110, 110, 150, 150,110,200,fill="yellow")
#绘制一个三角形,3个点分别为(110,110)、(150,150)、(110,200),填充色为黄色
line=c.create_line(250, 250, 350, 350, fill="white")
coord=250, 250,350,350
arc=c.create_arc(coord, start=0, extent=270, fill="red")
#绘制一个扇形,开始角度为 0°,旋转 270°,填充色为黄色

oval=c.create_oval(250, 50, 400, 100, fill="white")
#绘制一个椭圆,填充色为白色

line=c.create_line(50, 250, 100, 300, fill="white")
oval=c.create_oval(50, 250, 100, 300, fill="black")
#绘制一个圆,填充色为黑色

c.pack()
top.mainloop()
```

程序运行结果如图 11.7 所示。

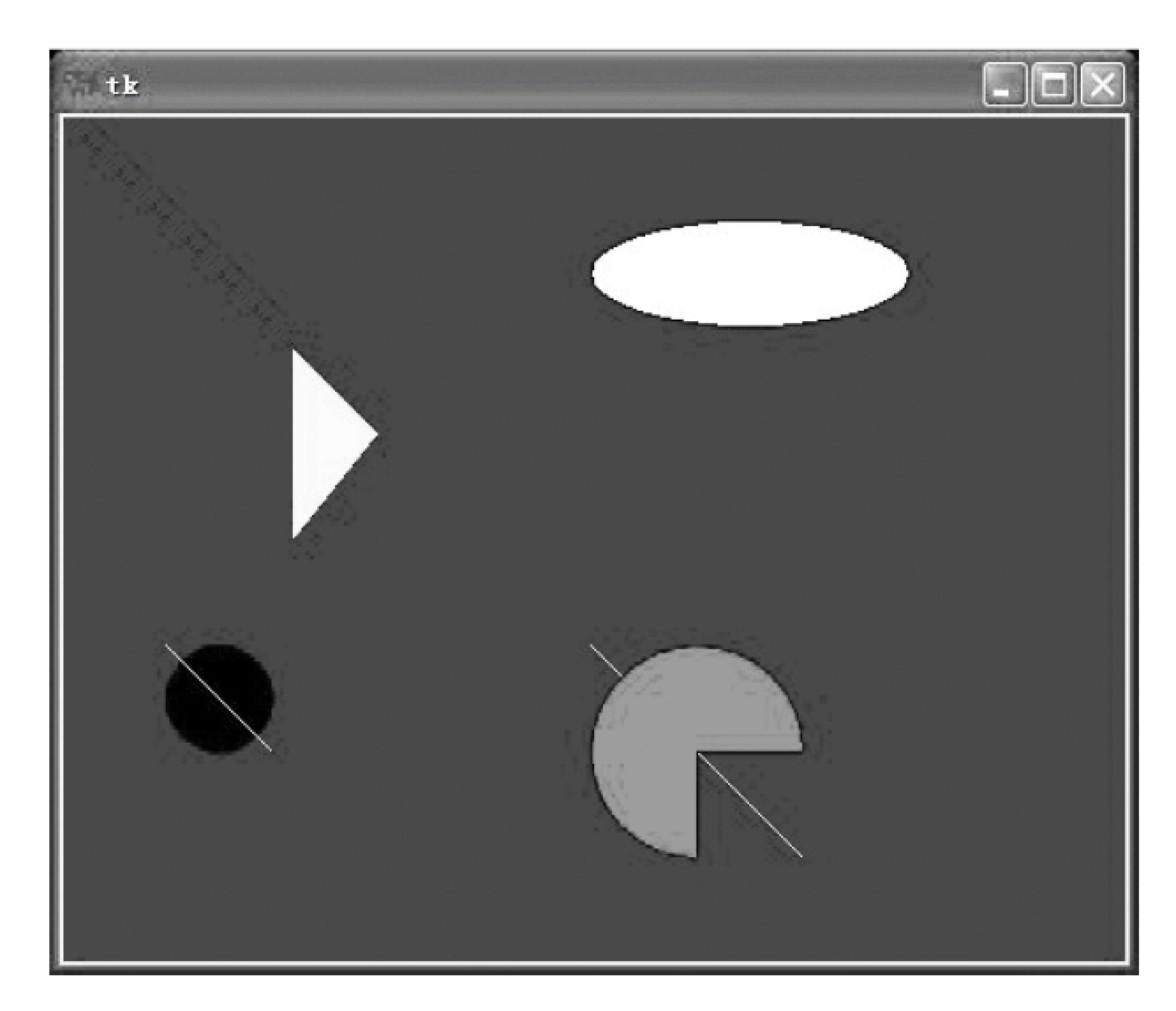


图 11.7 例 11-5 程序运行结果

11.4 Numpy 与 Matplotlib

11.4.1 Numpy 简介

NumPy(Numeric Python)是 Python 的开源数字扩展,用来存储和处理大型矩阵,比 Python 自身的嵌套列表结构高效。NumPy 提供了许多高级的数值编程工具,如矩阵数据类型、矢量处理等。Numpy 下载网址 http://sourceforge.net/projects/numpy/files,由于本书的 Python 是 32 位的 2.7.3 版本,故选择 numpy-1.7.0-win32-superpack-python2.7.exe 下载安装,如图 11.8 所示。

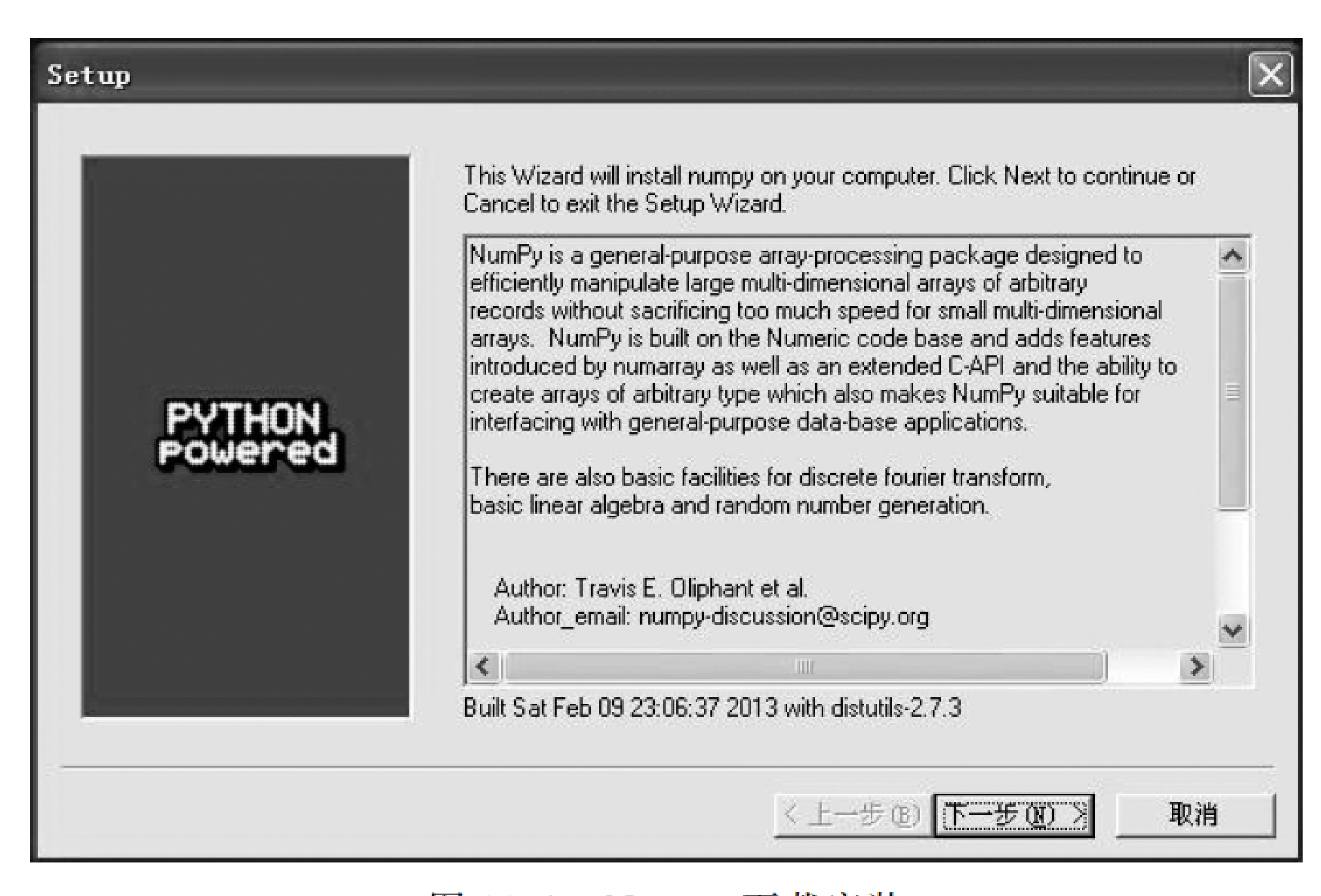


图 11.8 Numpy 下载安装

【例 11-6】 Numpy 举例。

```
>>>import numpy as np
>>>a=np.array([0,1,2,3,4,5])
>>>a
array([0, 1, 2, 3, 4, 5])
>>>a.ndim
1
>>>a.shape
(6,)
>>>b=a.reshape((3,2))
>>>b
array([[0, 1],
       [2, 3],
       [4, 5]])
>>>b.ndim
2
>>>b.shape
(3, 2)
>>>b[1][0]=10
>>>b
array([[ 0,  1],
       [10,  3],
       [ 4,  5]])
>>>a
array([ 0,  1, 10,  3,  4,  5])
>>>a * 2
array([ 0,  2, 20,  6,  8, 10])
```

11.4.2 Matplotlib 简介

Matplotlib 是 Python 二维绘图领域使用最广泛的图形框架,可以绘制多种形式的图形,包括线图、直方图、饼图、散点图以及误差线图等,较好地支持 TeX 排版命令。Matplotlib 类似于 MATLAB 和 R 语言,大部分函数与 Matlab 函数同名且使用方法一致。

Matplotlib 使用 numpy 模块提供的矩阵运算和各种数学函数,将 NumPy 统计计算结果可视化。Matplotlib 下载网址为 http://matplotlib.org/,如图 11.9 所示。选择与 Python 对应的版本,matplotlib-1.4.2.win32-py2.7.exe 下载安装,如图 11.10 所示。

Matplotlib 的学习可以从模仿例子开始,http://matplotlib.org/examples/index.html 网址提供了很多例子便于学习。

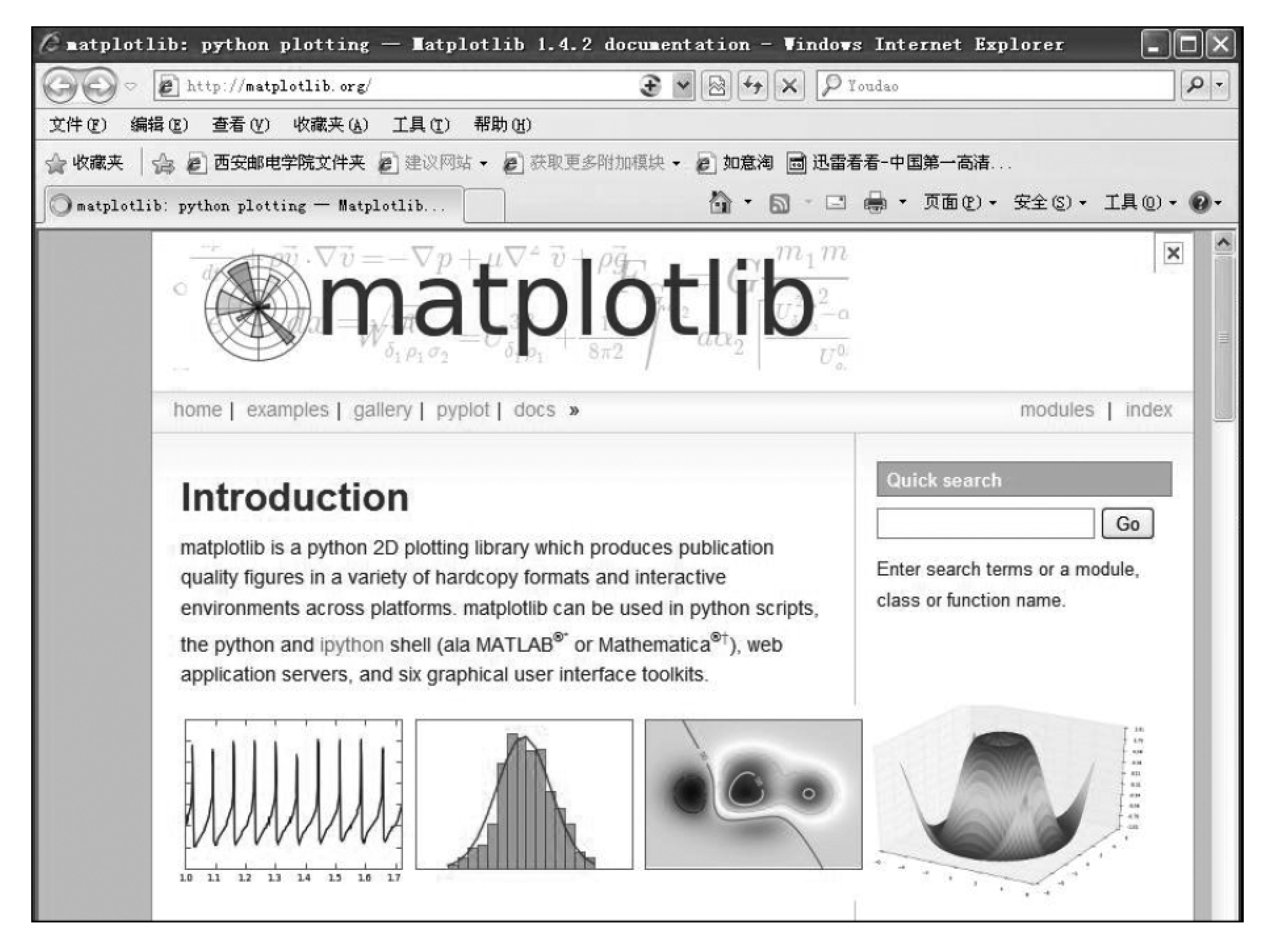


图 11.9　Matplotlib 网站



图 11.10　Matplotlib 下载安装

【例 11-7】 Matplotlib 举例。

```
import numpy as np
import matplotlib.pyplot as plt
x=np.linspace(0, 2 * np.pi)
offsets=np.linspace(0, 2*np.pi, 4, endpoint=False)
#Create array with shifted-sine curve along each column
```

```
yy=np.transpose([np.sin(x+phi) for phi in offsets])

plt.rc('lines', linewidth=4)
fig, (ax0, ax1)=plt.subplots(nrows=2)

plt.rc('axes', color_cycle=['r', 'g', 'b', 'y'])
ax0.plot(yy)
ax0.set_title('Set default color cycle to rgby')

ax1.set_color_cycle(['c', 'm', 'y', 'k'])
ax1.plot(yy)
ax1.set_title('Set axes color cycle to cmyk')

#Tweak spacing between subplots to prevent labels from overlapping
plt.subplots_adjust(hspace=0.3)
plt.show()
```

程序运行结果如图 11.11 所示。

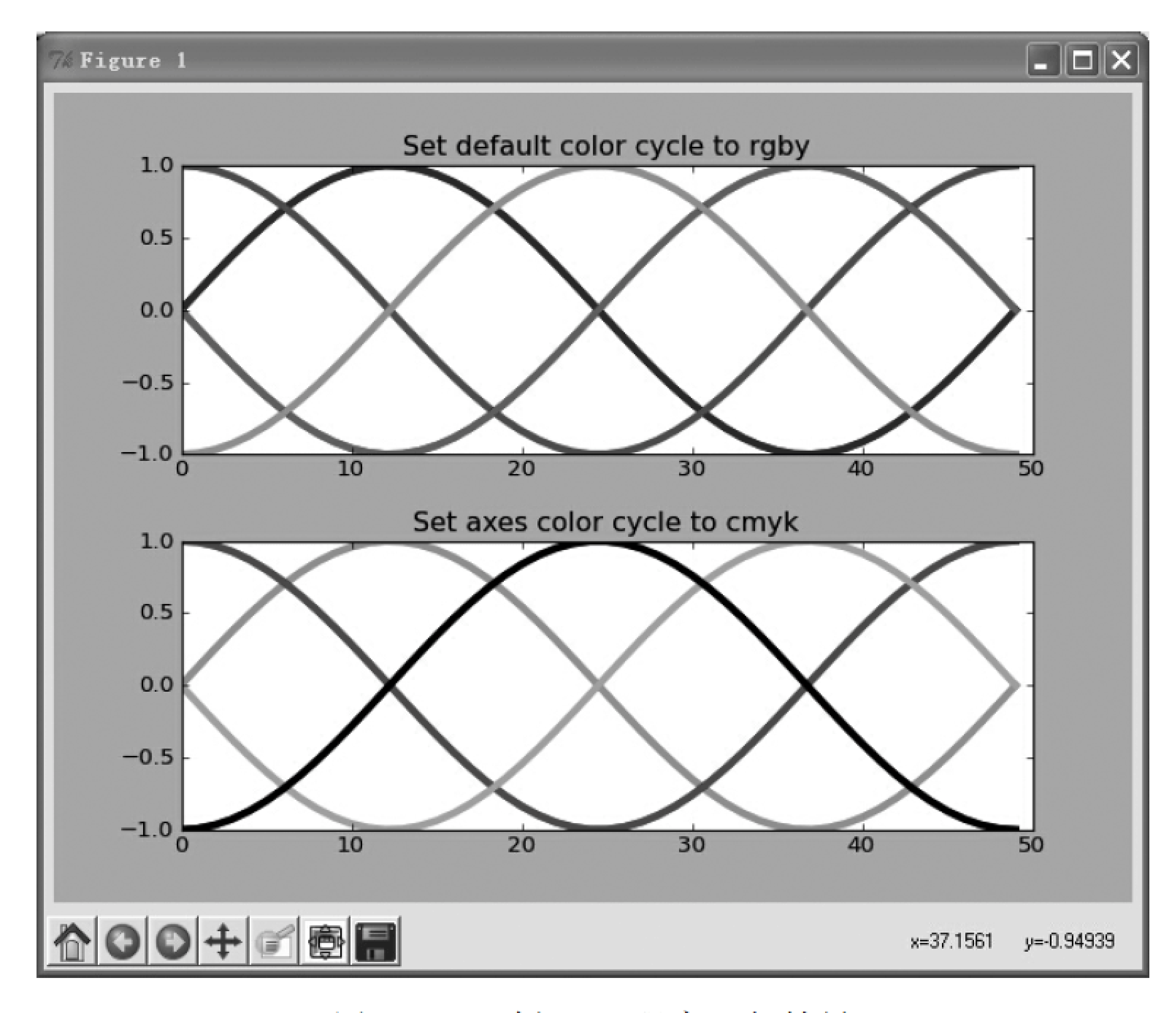


图 11.11 例 11-7 程序运行结果

11.5 习题

1. 学习 Turtule 绘图方法。
2. 学习 Canvas 绘图方法。
3. 学习 Numpy 和 Matplotlib 绘图方法。

第12章 数据库应用

数据库(Database)是按照数据结构来组织、存储和管理数据的仓库。本章首先介绍数据库的相关概念,如关系模型、结构化查询语言等知识。然后讲解 Python 访问数据库的两种方式,以及 Python 操作数据库的一般过程。最后,介绍 Python 如何在内置 SQLite3 数据库和 MySQL 数据库中存取数据。

12.1 数据库概念

12.1.1 数据库管理系统

相对文件系统而言,数据库管理系统为用户提供安全、高效、快速检索和修改的数据集合。由于数据库管理系统与应用程序文件分开独立存在,可为多个应用程序所使用,从而达到数据共享的目的。

数据库管理系统(Database Management System,DBMS),又称数据库,用于管理数据并提供数据库服务的软件,如 Access,SQL Server,Oracle 和 MySQL 等。

数据库管理系统具有如下功能:

(1) 数据库定义功能。使用数据定义语言(Data Definition Language,DDL)生成和描述各种数据对象。

(2) 数据库操纵功能。使用数据操作语言(Data Manipulation Language,DML)对数据库进行各种数据操作(检索、插入、删除和更新等)。

(3) 数据库运行控制功能。用于数据库的安全性、完整性检查,以及数据共享的并发控制等,目的是保证数据库的可用性和可靠性。

简单地说,数据库就是按照数据结构来组织、存储和管理数据的仓库。例如,人事部门将职工的基本情况(如职工号、姓名、年龄、性别、籍贯、工资和简历等)存放在一张表中。这张表就可以看作一个数据库,可以根据需要随时查询某职工的基本情况,也可以查询工资在某个范围内的职工人数等信息。

12.1.2 关系型数据库

一个典型的关系型数据库通常由一个或多个被称为表格的对象组成。数据库中的所有数据或信息都被保存在这些数据库表格中。表格由行和列组成,其中每一列包括了该

列名称、数据类型以及列的其他属性等信息，而行则具体包含某一列的记录或数据。例如，某学校的学生关系就是一个二元关系，如图 12.1 所示。

name	sex	qq	tele	school
周黎明	男	9828322	88765238	西安交通大学
何明明	女	8876542	99887645	山西师范大学

图 12.1　关系型数据库的表结构

作为一个关系的二维表，必须满足以下条件：

(1) 表中每一列必须是基本数据项(即不可再分解)。

(2) 表中每一列必须具有相同的数据类型(如字符型或数值型)。

(3) 表中每一列的名字必须是唯一的。

(4) 表中不应有内容完全相同的行。

(5) 行的顺序与列的顺序不影响表格中所表示的信息的含义。

当前流行的数据库(如 MySQL 等)都是基于关系模型的关系数据库管理系统。关系模型认为世界由实体(Entity)和联系(Relationship)构成。实体是相互可以区别，具有一定属性的对象。联系是指实体之间的关系，一般分以下三种类型：

(1) 一对一(1∶1)：实体集 A 中每个实体至多只与实体集 B 中一个实体相联系。反之亦然。例如，班级和班主任的关系，如图 12.2 所示。

(2) 一对多(1∶n)：实体集 A 中每个实体与实体集 B 中多个实体相联系，而实体集 B 中每个实体至多只与实体集 A 中一个实体相联系。例如，学生和班级的关系，如图 12.3 所示。

(3) 多对多(m∶n)：实体集 A 中每个实体与实体集 B 中多个实体相联系，反之，实体集 B 中每个实体也与实体集 A 中多个实体相联系。例如，学生和课程的关系，如图 12.4 所示。

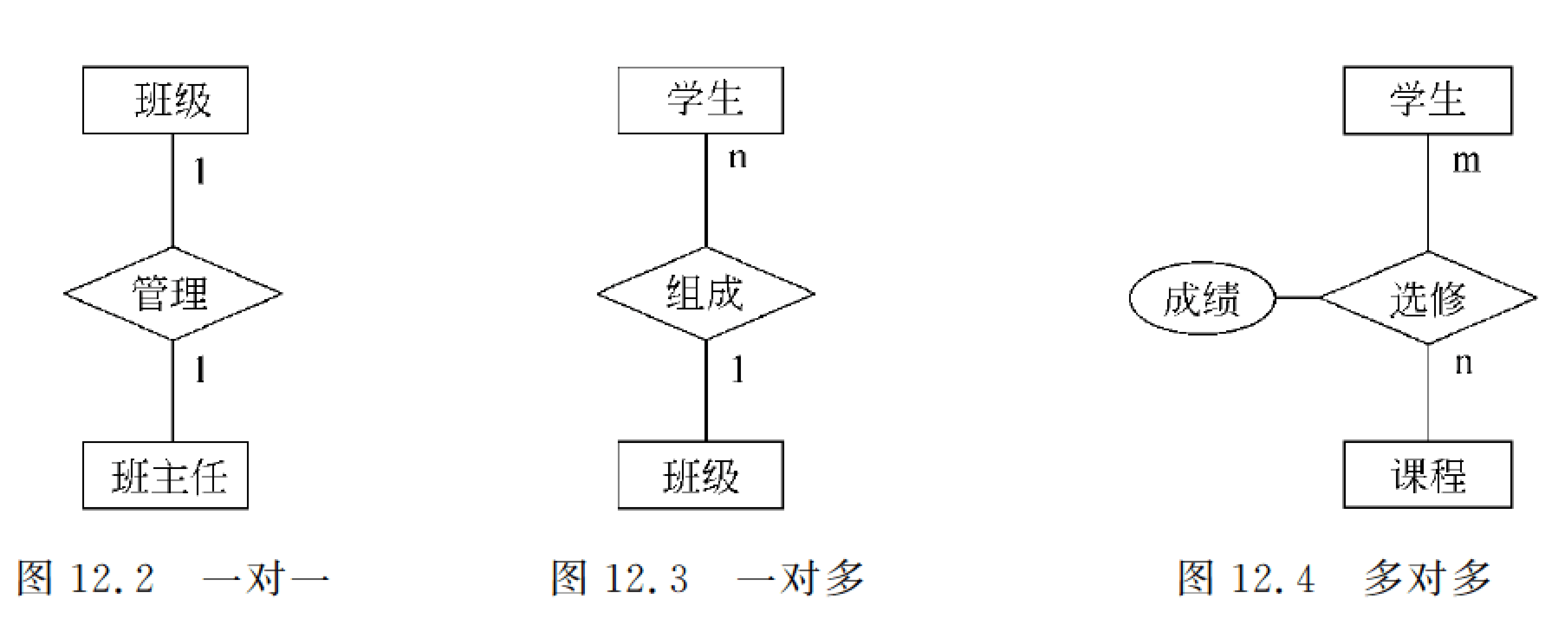


图 12.2　一对一　　图 12.3　一对多　　图 12.4　多对多

12.1.3　结构化查询语言

结构化查询语言(Structured Query Language，SQL)是操作数据库的工业标准语言，作为通用的、专门操作数据库的语言，SQL 可以确切指定想要检索的记录以及按什么顺序检索。

下面介绍关于 SQL 语言的四条命令。

(1) select 语句

从数据库中获取符合查询条件的数据。

select 语法如下所示：

```
select 字段表 from 表名 where 查询条件 group by 分组字段 order by 字段 [Asc|Desc]
```

【例 12-1】 在 employee 表查找名字为“May”雇员的信息。

```
select *  from  employee  where  name= "May";
```

(2) update 语句

update 创建一个更新查询来按照某个条件修改特定表中的字段值。其语法如下：

```
update [表集合]  set [表达式]  where [条件]
```

【例 12-2】 将 employee 表中性别为“女”的雇员的年龄增加一岁。

```
update  employee  set  age=age+1 where sex= "女";
```

(3) delete 语句

删除 from 子句中列出的、满足 where 子句的一个或多个表中的记录。

delete 语法如下所示：

```
delete [表字段]  from [表集合]  where [条件]
```

【例 12-3】 将 employee 表中名字为 May 的雇员删除。

```
Delete  from  employee    where  name= "May";
```

(4) insert 语句

向表中添加一条记录。

insert 语句语法如下所示：

```
insert into 数据表名 (字段名 1,字段名 2,…)  values(数据 1,数据 2,…)
```

【例 12-4】 在 employee 表中添加一条记录。

```
insert into employee(name,age,address,city) values("May",45,"西安邮电大学",
"西安");
```

结构化查询语言如表 12.1 所示。

表 12.1 结构化查询语言

常用 SQL 命令	描　　述
insert	往数据库表中插入记录
delete	从数据库表中删除记录
select	在数据库中查找满足特定条件的记录
update	改变特定记录和字段的值

续表

常用 SQL 命令子句	描　　述
from	用来为从其中选定记录的表命名
where	用来指定所选记录必须满足的条件
group by	用来把选定的记录分成特定的组
having	用来说明每个组需要满足的条件
order by	用来按特定的次序将记录排序
合计函数	**描　　述**
avg	用来获得特定字段中的值的平均数
count	用来返回选定记录的个数
sum	用来返回特定字段中所有值的总和
max	用来返回指定字段中的最大值
min	用来返回指定字段中的最小值

12.2　Python 数据库访问模块

针对 MySQL，Oracle 和 DB2 等数据库，Python 提供了通用数据库访问模块以及专用数据库访问模块访问各个数据库。

12.2.1　通用数据库访问模块

通用数据库访问模块有 ODBC 和 JDBC 两种。

（1）ODBC

开放数据库互连（Open Database Connectivity，ODBC）是微软公司开放服务结构中有关数据库的一个组成部分，它建立了一组规范，并提供了一组对数据库访问的标准 API（应用程序编程接口）。这些 API 利用 SQL 来完成其大部分任务。ODBC 本身也提供了对 SQL 语言的支持，用户可以直接将 SQL 语句送给 ODBC。开放数据库互连（ODBC）是 Microsoft 提出的数据库访问接口标准。Python 提供的通过 ODBC 访问数据库的模块有 ODBC Interface 和 Pyodbc 等。

（2）JDBC

JDBC（Java Data Base Connectivity）是 Java 用于执行 SQL 语句的 API，可为多种关系数据库提供统一访问，它由一组用 Java 语言编写的类和接口组成。JDBC 提供了一种基准用于构建更高级的工具和接口，使数据库开发人员能够编写数据库应用程序。Python 提供了通过 JDBC 访问数据库的模块，即 zxJDBC 模块。

12.2.2　专用数据库访问模块

对于不同的数据库，Python 也提供了专用数据库访问模块，如表 12.2 所示。

表 12.2 专用数据库访问模块

数 据 库	Python 模块	网 址
MySQL	MySQLdb	http://sourceforge.net/projects/Mysql-python
PostgreSQL	PyGreSQL	http://www.pygresql.org
Oracle	DCOracle2	http://www.zope.org/Members/matt/dco2
IBM DB2	Pydb2	http://sourceforge.net/projects/pydb2
SQL Server	pymssql	http://pymssql.sourceforge.net

12.3 Python 操作数据库

12.3.1 连接对象和游标

Python 操作数据库，有两个重要的概念，分别是数据库连接对象和游标。

(1) 数据库连接对象

打开数据库时返回 conn 对象，是数据库连接对象(connect)，具有以下操作：

```
conn=sqlite3.connect(host,user,passwd,db)
```

参数如下所示：

- host，连接的数据库服务器主机名，默认为本地主机(localhost)。
- user，连接数据库的用户名，默认为当前用户。
- passwd，连接密码，没有默认值。
- db，连接的数据库名，没有默认值。

Conn 对象具有如下方法：

- commit()，事务提交。
- rollback()，事务回滚。
- close()，关闭一个数据库连接。

(2) 游标

游标(cursor)是数据库管理系统为用户开设的一个数据缓冲区，存放 SQL 语句的执行结果，每个游标区都有一个名字，用户可以用 SQL 语句逐一从游标中获取记录，进行操作处理。

定义一个游标如下所示：

```
cu=conn.cursor()            #cu为一变量
```

游标对象有如下操作：

- execute()，执行 sql 语句。
- executemany，执行多条 sql 语句。
- fetchone()，从结果中取一条记录，并将游标指向下一条记录。

- fetchmany()，从结果中取多条记录。
- fetchall()，从结果中取出所有记录。
- scroll()，游标滚动。
- close()，关闭游标。

12.3.2　操作数据库过程

Python 操作数据库过程如下所示：

步骤 1：用 db.connect 创建数据库连接，返回连接对象为 conn。

步骤 2：操作数据库中记录。

① 用 conn.cursor 创建游标对象 cu。

② 通过 cu.execute 查询数据库，用 cu.fetchall，cu.fetchone 和 cu.fetchmany 等方法返回查询结果。

③ 用 conn.commit 修改数据库。

步骤 3：关闭 cu，conn。

12.4　Python 与两个数据库

12.4.1　SQLite3

SQLite3 数据库是一款非常小巧的嵌入式开源数据库软件，也就是说没有独立的维护进程，所有的维护都来自于程序本身。SQLite3 支持 Windows，Linux 和 UNIX 等主流操作系统，同时与众多程序设计语言配合使用，例如 Tcl，C#，PHP 和 Java 等，SQLite3 相对于 MySQL 和 PostgreSQL 等开源数据库管理系统，具有处理速度较快的特点。

Python2.5 之后内置了 SQLite3，导入即可。语法如下所示：

```
Import sqlite3
```

【例 12-5】　Python 操作 SQLite3。

```
import sqlite3
#调用 connect 函数指定数据库,如果不存在就新创建后再打开
conn=sqlite3.connect("d:/stud.db")          #连接 d:/stud.db 数据库
cu=conn.cursor()

#如果存在表先删除
sql_del="drop table if exists student"
try:
    cu.execute(sql_del)
except sqlite3.Error,e:
    print "fail!!","\n",e.args[0]

cu.execute("create table student(id integer primary key,name char(12) UNIQUE)")
```

```
#创建 student 表,student 表有 id 和 name 两个字段,其中 id 为主键
cu.execute("""insert into student(id,name) values("1","zhou")""")
cu.execute("""insert into student(id,name) values("2","wang")""")
#执行两条插入语句
sql_select="SELECT * FROM student;"
cu.execute(sql_select)
#从 student 表中检索所有数据
li=cu.fetchall()
for row in li:
    if type(row)!=unicode:
        print str(row)

#将检索的记录输出到屏幕上
conn.commit()
conn.close()
```

输出如图 12.5 所示。

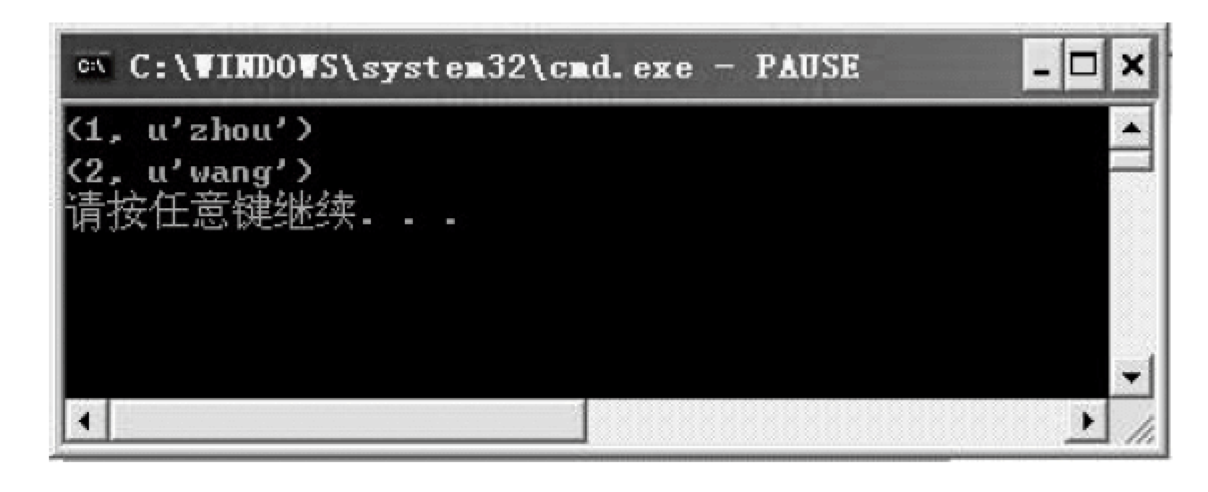


图 12.5　例 12-5 程序运行结果

12.4.2　MySQL

MySQL 是当今最流行的关系型数据库管理系统,作为开源软件,MySQL 体积小,采用关联数据库将数据保存在不同的表中,而不是将所有数据放在一个大仓库内,具有较快的存取速度和较高的灵活性。

Python 操作 MySQL 数据库需要使用 MySQLdb 接口,网址是 http://sourceforge.net/projects/mysql-python/ ,下载与 Python2.7.3 版本相应的 MySQL-python-1.2.4b4.win32-py2.7 文件,双击安装,默认安装路径为 Python 安装路径,c:\Python27,安装如图 12.6 所示。

安装成功后,启动 Python,输入 import MySQLdb 导入 MySQLdb。

```
>>> import MySQLdb
>>> conn=MySQLdb.connect(host='localhost',user='root',passwd='root',db=
'test',port=3306)
#主机,用户,数据库名称,端口号(MySQL 使用 3306)
```

Python 对于 MySQL 的操作和对 SQLite3 操作流程完全相同。

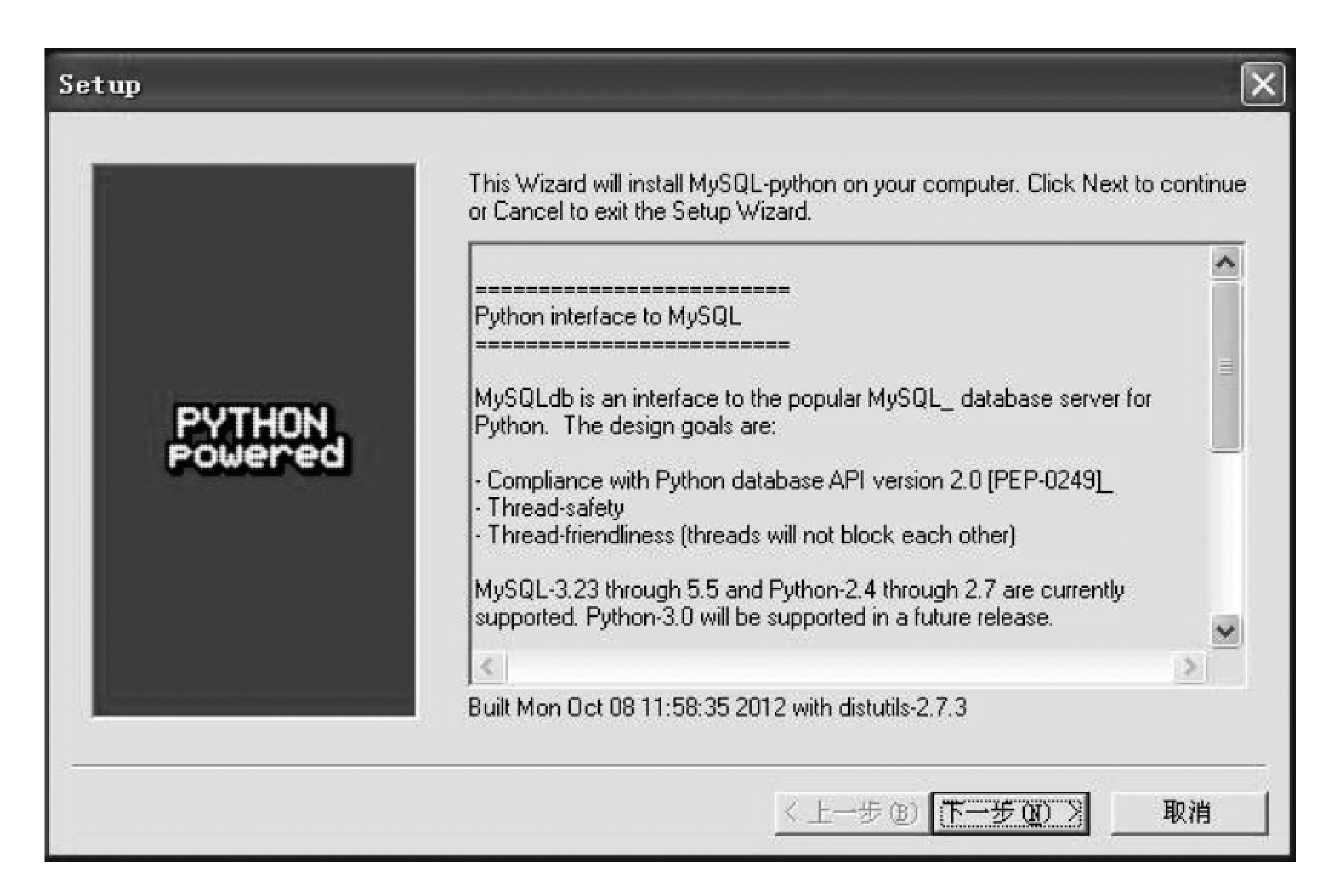


图 12.6　MySQL 对比安装步骤

12.5　习题

1. 什么是关系数据库？如何在 Access 中建立数据库，并进行增删查改操作？
2. SQL 语句有哪些？分别如何使用？
3. Python 如何实现操作数据库？
4. 实现例 12-5 Python 操作 SQLite3。
5. 下载 MySQL，实现例 12-5 功能。

第13章 网络编程

本章首先介绍网络编程的基本知识，TCP/IP 四层模型等相关内容。然后介绍 socket 编程的相关知识，最后讲解如何通过 SMTP 和 POP3 协议用 Python 实现发送邮件和收取邮件。

13.1 网络基础知识

计算机网络是指将地理位置不同的具有独立功能的多台计算机及其外部设备，通过通信线路连接起来，在网络操作系统、网络管理软件及网络通信协议的管理和协调下，实现资源共享和信息传递的计算机系统。

网络编程就是如何在程序中实现通过网络协议与其他计算机进行通信。网络编程模型为客户机/服务器(Client/Server，C/S)结构，客户机申请服务，服务器响应服务。网络编程模型如图 13.1 所示。

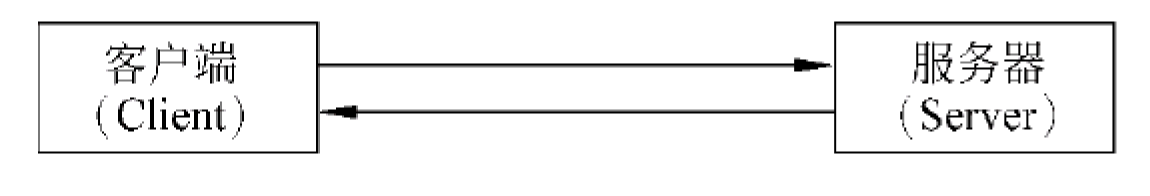


图 13.1 客户机/服务器结构

13.2 TCP/IP

为了把全世界不同类型的计算机连接起来，必须规定一套全球通用的协议，其中最重要的就是互联网协议簇(Internet Protocol Suite)。互联网协议簇是一种计算机网络模型以及运行在互联网和类似网络上的通信协议的集合，它一般被称为 TCP/IP，原因就是 TCP(Transmission Control Protocol)和 IP(Internet Protocol)是互联网协议簇中最重要的两个协议，也是在标准中最早被定义的协议。IP 负责把数据分割成 IP 包，从一台计算机通过网络发送到另一台计算机，TCP 建立在 IP 之上，负责在两台计算机之间建立可靠连接，保证数据包顺利发送与接收。TCP 通过握手建立连接后，对每个 IP 包进行编号发送，如果 IP 包发生丢掉情况，就自动重发，保证数据的可靠安全性。

13.2.1　TCP/IP 四层模型

TCP/IP 是一组用于实现网络互连的通信协议，其参考模型有网络接入层、网际互联层、传输层和应用层四个层次。

1. 应用层

应用层对应于 OSI 参考模型的应用层、表示层和会话层，为用户提供各种服务。

2. 传输层

传输层对应于 OSI 参考模型的传输层，为应用层实体提供端到端的通信功能，保证了数据包的顺序传送及数据的完整性。该层定义了 TCP 和 UDP 两个主要的协议。

（1）传输控制协议（Transmission Control Protocol，TCP）

TCP 提供的是一种可靠的、通过“三次握手”来连接的数据传输服务，TCP 提供可靠通信服务，例如 HTTP，FTP 和 SMTP 等都是使用 TCP 传输协议。

（2）用户数据报协议（User Datagram Protocol，UDP）

UDP 提供的则是不保证可靠、无连接的数据传输服务，但其传输更简单高效，适于实时交互性应用，如音频、视频会议等。

3. 网际互联层

网际互联层对应于 OSI 参考模型的网络层，主要解决主机到主机的通信问题，包括网际协议（IP），IP 是网际互联层最重要的协议，提供可靠、无连接的数据报传递服务。

4. 网络接入层

网络接入层又称为网络访问层，与 OSI 参考模型中的物理层和数据链路层相对应，负责监视数据在主机和网络之间的交换。

OSI 参考模型与 TCP/IP 四层模型如图 13.2 所示。

OSI参考模型	TCP/IP四层模型
应用层	应用层
表示层	
会话层	
传输层	传输层
网络层	网际互联层
数据链路层	网络接入层
物理层	

图 13.2　OSI 参考模型与 TCP/IP 四层模型

13.2.2　IP 地址和端口号

网络通信必须有 IP 地址和端口号两个重要的信息才能完成数据的传输。

（1）IP 地址

为了区分网络中的每一台主机，Internet 采用一种全局通用的地址格式，为网络中的每一台主机分配唯一的地址（IP 地址）。IP 地址是通信主机的唯一标识地址，相当于邮件通信中双方的邮件地址。常见的 IP 地址分为 IPv4 与 IPv6 两大类。IPv4 地址是一个 32 位的二进制数，通常被分割为 4 个“8 位二进制数”（也就是 4 个字节）。IPv4 地址通常用“点分十进制”表示成（a. b. c. d）的

形式,其中,a,b,c,d 都是 0～255 之间的十进制整数。例如 192.168.1.1,其中以 127 开头的 IP 地址(如 127.0.0.1)为本机回送地址,用于网络软件测试及本地机进程间通信。由于互联网的蓬勃发展,IP 地址的需求量越来越大,IPv4 的地址只有 32 位,已经使用殆尽。因此,提出了 IPv6,IPv6 采用 128 位地址长度,重新定义了 IP 地址空间,大大提高了互联网的服务水平。由于 IP 地址基于数字标识,不易记忆,因此使用域名(Domain Name System,DNS)帮助记忆,如 www.sina.com.cn 就是新浪网的域名。

(2) 端口号

同一台计算机可以运行多个网络程序,如 QQ、IE 浏览器、MySQL 数据库等,每个网络程序都有唯一的端口号,端口号的作用就是用于区分不同的网络应用程序,相当于邮件通信中的收件人姓名。例如,新浪网的端口号为 80 端口。

注意: 80 端口是 Web 服务的标准端口。其他服务都有相对应的标准端口号,例如 SMTP 服务是 25 端口,FTP 服务是 21 端口等。另外,端口号小于 1024 的是 Internet 标准服务的端口,端口号大于 1024,用户可以任意使用。

13.3 Socket

Socket 作为网络编程的一个抽象概念,用于描述 IP 地址和端口,表示"打开了一个网络链接"。一个 Socket 绑定到一个端口上,不同的端口对应于不同的服务。因此,基于 Socket 编程需要知道目标计算机的 IP 地址、端口号以及协议类型等,按照协议类型的不同,基于 Socket 的编程分为 TCP 连接和 UDP 连接。

13.3.1 TCP 连接

TCP 连接分为服务器连接和客户端连接。

(1) 服务器连接

建立服务器连接需要 6 个步骤,如图 13.3 所示。

第 1 步,创建 socket 对象。调用 socket 构造函数。

```
socket=socket.socket(family,type)
```

参数说明:

family 的值可以是 AF_UNIX(UNIX 域,用于同一台机器上的进程间通信),也可以是 AF_INET(对于 IPV4 协议的 TCP 和 UDP),至于 type 参数,可以是 SOCK_STREAM(流套接字)或者 SOCK_DGRAM(数据报文套接字),以及 SOCK_RAW(raw 套接字)。

第 2 步,通过 socket 的 bind 方法绑定到指定地址上。

```
socket.bind(address)
```

address 必须是一个双元素元组(host,port),其中 host 是主机名或者 ip 地址,port 为端口号。

第 3 步,通过 socket 的 listen 方法接收连接请求。

```
socket.listen(backlog)
```

参数说明：

backlog 指定了最多连接数，至少为 1。

第 4 步，通过 socket 的 accept 方法等待客户请求一个连接。

```
connection,address=socket.accept()
```

第 5 步，处理阶段，服务器和客户通过 send 和 recv 方法传输数据。

服务器调用 send，并采用字符串形式向客户发送信息。send 方法返回已发送的字符个数。服务器使用 recv 方法从客户接收信息。调用 recv 时，必须指定一个整数来控制本次调用所接受的最大数据量。

第 6 步，传输结束，服务器调用 socket 的 close 方法关闭连接。

(2) 客户端连接

建立一个简单客户连接则需要 4 个步骤，如图 13.4 所示。

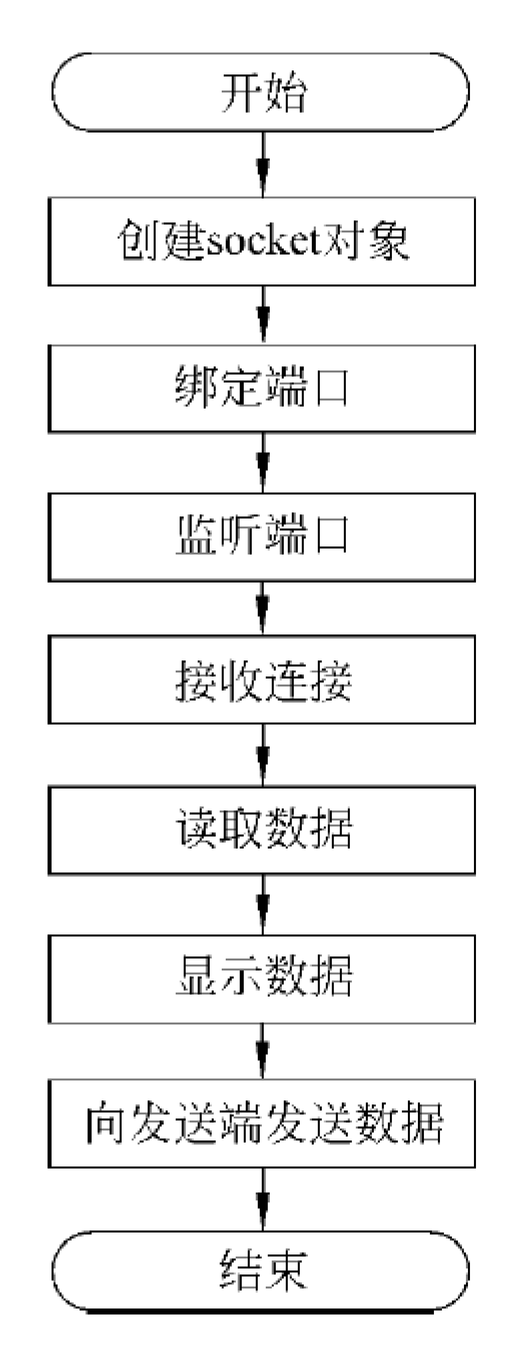


图 13.3　服务器连接流程

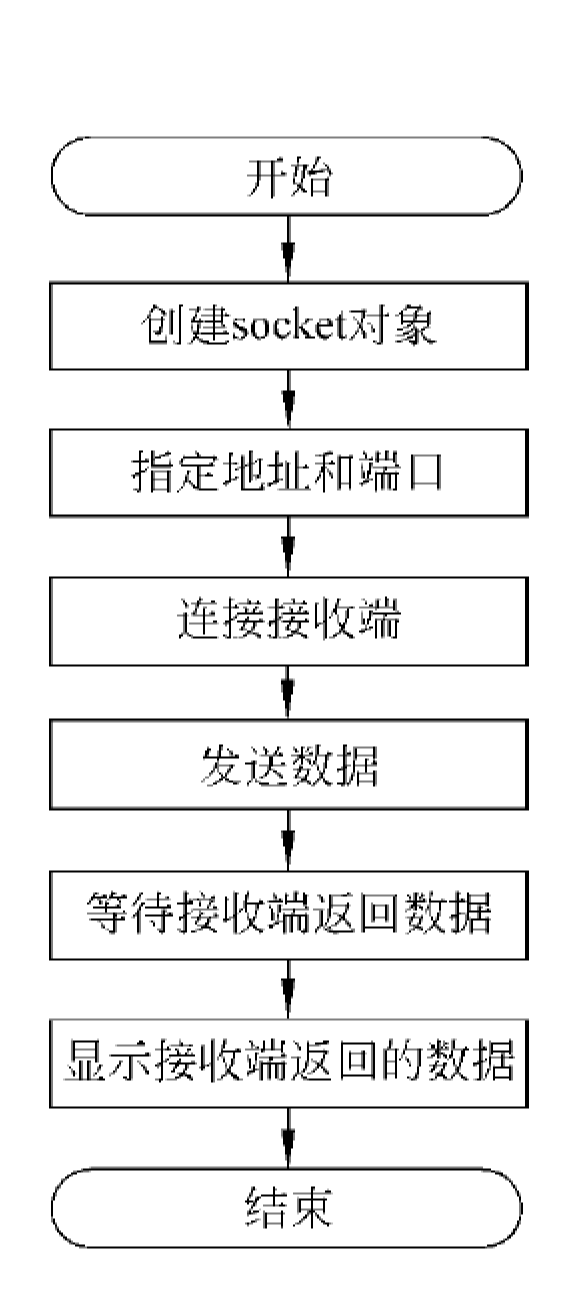


图 13.4　客户端连接流程

第 1 步，创建一个 socket 以连接服务器。

```
socket=socket.socket(family,type)
```

第 2 步，使用 socket 的 connect 方法连接服务器。

```
socket.connect((host,port))
```

参数说明：

host 代表服务器主机名或 IP，port 代表服务器进程所绑定的端口号。

第 3 步，客户和服务器通过 send 和 recv 方法通信。

第 4 步，客户通过调用 socket 的 close 方法关闭连接。

【例 13-1】 基于 Socket 的 TCP 连接。

Server. py： #服务器

```
import socket
serversocket=socket.socket(socket.AF_INET, socket.SOCK_STREAM)
serversocket.bind(('127.0.0.1', 8000))
serversocket.listen(5)
print 'Waiting for connection...'
clientsocket, addr=serversocket.accept()
data=clientsocket.recv(1024)
print "received message:",repr(data)
clientsocket.send('Hello, %s!' %data)
clientsocket.close()
serversocket.close()
```

Client. py：

```
import socket
clientsocket=socket.socket(socket.AF_INET, socket.SOCK_STREAM)
clientsocket.connect(('127.0.0.1', 8000))
clientsocket.send('hello')
data=clientsocket.recv(1024)
print 'received sendback:',data
clientsocket.close()
```

13. 3. 2 UDP 连接

基于 Socket 的 UDP 连接也分为服务器连接和客户端连接。

(1) 客户端连接

客户端连接如图 13. 5 所示，具有如下步骤：

步骤 1：创建 Socket 对象。

步骤 2：通过 sendto 方法传输数据。

步骤 3：传输结束，调用 close 方法关闭连接。

(2) 服务器连接。

服务器连接如图 13. 6 所示，具有如下步骤：

步骤 1：创建 Socket 对象。

步骤 2：将 Socket 绑定到指定地址。

步骤 3：通过 recvfrom 方法传输数据。

步骤 4：传输结束，调用 close 方法关闭连接。

【例 13-2】 基于 UDP 的 Socket 连接。

服务器 server. py：

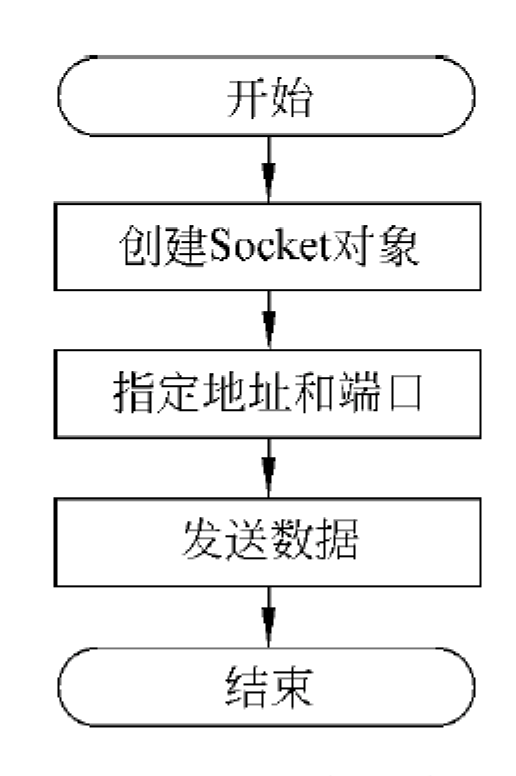


图 13.5 客户端连接流程

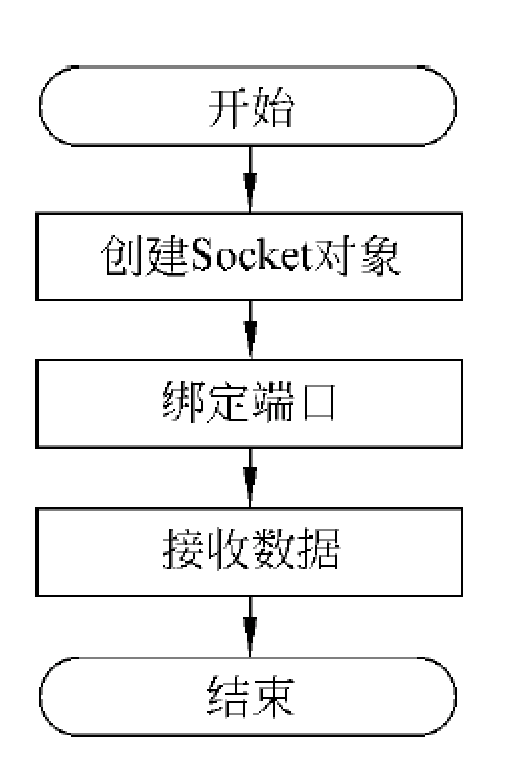


图 13.6 服务器连接流程

```
import socket
s=socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
s.bind(("127.0.0.1", 5005))
data, addr=s.recvfrom(1024)
print ' received message:%s' %data
s.close()
```

客户机 Client. py:

```
import socket
s=socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
data='hello,server'
print data
s.sendto(data , ("127.0.0.1" ,5005))
s.close()
```

首先运行 server. py，服务器处于等待接收，其后运行 client. py，程序运行结果如图 13.7 所示。



图 13.7 例 13-2 程序运行结果

13.4 电子邮件

13.4.1 SMTP 发送邮件

SMTP(Simple Mail Transfer Protocol，简单邮件传输协议)用于发送邮件，Python

内置对 SMTP 的支持,可以发送纯文本邮件、HTML 邮件以及带附件的邮件。Python 提供了 smtplib 模块表示与 SMTP 服务器之间的连接,通过这个连接可以向 smtp 服务器发送指令,执行相关操作(如登录、发送邮件)。

```
smtplib.SMTP([host[, port[, local_hostname[, timeout]]]])
```

smtplib. SMTP 提供如下方法:

- connect([host[,port]]):连接到指定的 smtp 服务器。
- login(user,password) :登录到 smtp 服务器。
- sendmail(from_addr,to_addr,msg):发送邮件。
- quit():断开与 smtp 服务器的连接。

【例 13-3】 SMTP 发送邮件。

```
import smtplib
from_mail='pangshengli@ yeah.net'
to_mail='zhouyuanzhe@ 163.com'
server=smtplib.SMTP('smtp.yeah.net')
server.login('pangshengli@ yeah.net','xxxxxx')
msg='''From: pangshengli@ yeah.net\r\nTo: zhouyuanzhe@ 163.com\r\nSubject: this is a
Email from python demo\r\n\r\nJust for test~_~'周老师好,我在做 SMTP 测试,如果你收到了说
明工作正常! '''
server.sendmail(from_mail,to_mail,msg)
server.quit()
```

13.4.2 POP3 收取邮件

Python 内置的 poplib 模块实现了 POP3 协议,用来接收邮件。Python 的 poplib 收取邮件的过程一般具有如下步骤:

(1) 连接 pop3 服务器 (poplib. POP3)

(2) 发送用户名和密码进行验证 (poplib. POP3. user,poplib. POP3. pass_)

(3) 获取邮箱中信件信息 (poplib. POP3. stat)

(4) 收取邮件 (poplib. POP3. retr)

(5) 退出 (poplib. POP3. quit)

【例 13-4】 POP3 收取邮件。

```
import poplib
#连接 pop3 服务器
emailServer=poplib.POP3('pop3.163.com')
#用户名
emailServer.user(' ****@163.com')
#密码
emailServer.pass_('**** * ')
#设置调试模式,查看与服务器交互信息
emailServer.set_debuglevel(1)
```

```
#获取服务器上信件信息,emailMsgNum为邮件数量,emailSize为字节总数
emailMsgNum, emailSize=emailServer.stat()
print 'email number is %d and size is %d'%(emailMsgNum, emailSize)
#取第一封邮件完整信息,返回值按行存储在content[1]列表
content=emailServer.retr(1)
print 'lines' ,len(content)
for line in content[1]:
    print line
#退出
emailServer.quit()
```

程序运行结果如图 13.8 所示。

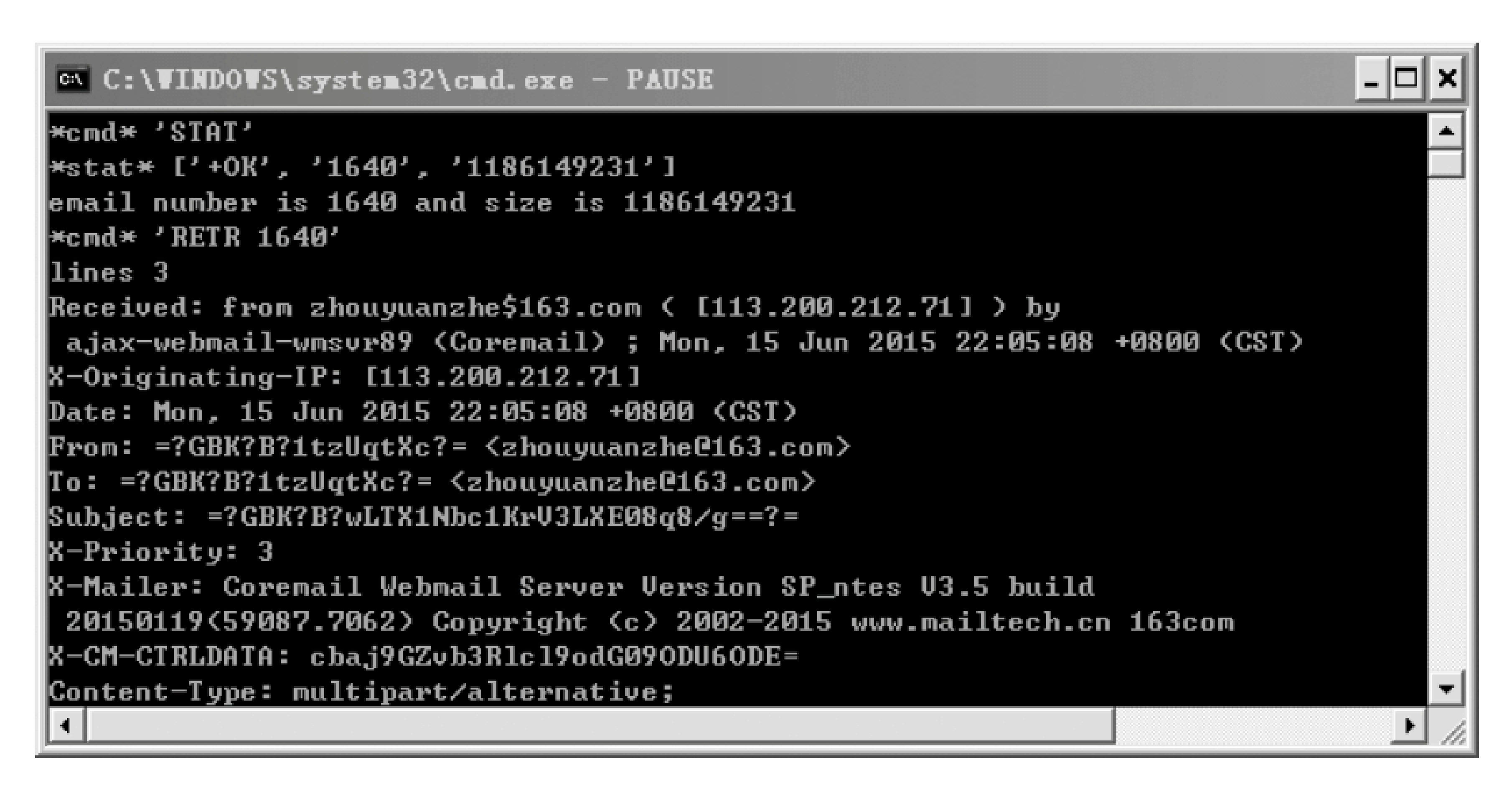
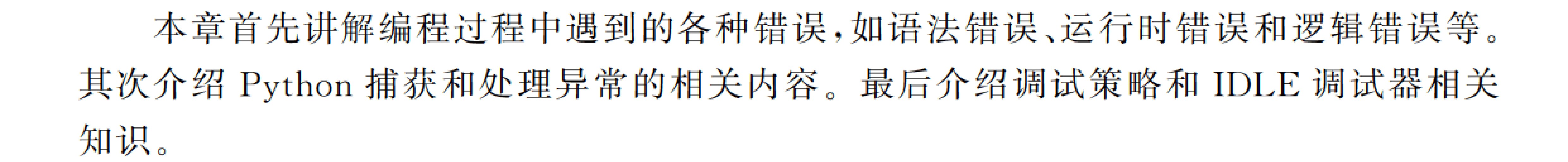


图 13.8　程序运行结果

13.5　习题

1. TCP/IP 协议的四层模型是什么?
2. 如何理解 IP 地址?
3. 端口号的作用是什么?
4. 实现教材中基于 TCP 的 Socket 连接的举例。
5. 实现教材中基于 UDP 的 Socket 连接的举例。
6. 实现 SMTP 发送邮件和 POP3 收取邮件举例。

第14章 异常处理



本章首先讲解编程过程中遇到的各种错误，如语法错误、运行时错误和逻辑错误等。其次介绍 Python 捕获和处理异常的相关内容。最后介绍调试策略和 IDLE 调试器相关知识。

14.1 错误类型

计算机编程过程中出现的错误大致分为语法错误、运行时错误和逻辑错误等。下面依次进行介绍。

14.1.1 语法错误

在编辑代码时，Python 会对输入的代码直接进行语法检查，例如，print 之前多了空格或者按了 Tab 键，都会导致在 Python-Shell 里运行语句时出现错误。

语法错误较容易发现和改正，Python 不但会给出语法错误提示，而且会输出错误位置。

【例 14-1】 语法错误举例。

```
>>>Print 'Hello World'
    File "<stdin>", line 1
        Print 'Hello World'
SyntaxError: invalid syntax
>>>print 'Hello World'
Hello World
```

【解析】 print 误拼为 Print，大小写错误会引发语法错误。

14.1.2 运行时错误

有些代码在编写时没有错误，但在程序运行过程中发生异常，这类错误称为“运行时错误”。例如，执行除数为零的除法运算、打开不存在的文件、数据类型不匹配、列表索引越界等。

【例 14-2】 运行时错误举例。

```
>>> f=open("a.txt")

Traceback (most recent call last):
    File "<pyshell#0>", line 1, in<module>
        f=open("a.txt")
IOError: [Errno 2] No such file or directory: 'a.txt'
```

【解析】 a. txt 文件不存在，引起运行时错误。

14.1.3 逻辑错误

逻辑错误又称为语义错误，表现形式是程序运行时不报错，但结果不正确，这往往是由于程序存在逻辑上的缺陷。例如，运算符使用的不合理、语句的次序不对等。对于逻辑错误，Python 解释器无能为力，只能由人工发现。

【例 14-3】 逻辑错误举例。

```
>>> import math
>>> a=1;b=2;c=1
>>> x1=-b+math.sqrt(b*b-4*a*c)/2*a
>>> x2=-b-math.sqrt(b*b-4*a*c)/2*a
>>> print x1,x2
-2.0 -2.0
```

【解析】 一元二次方程求根公式有误导致的逻辑错误。

14.2 捕获和处理异常

异常(Exception)是因程序的例外、违例、出错等情况而在正常控制流以外采取的行为。异常一般分为如下两个阶段：

(1) 异常发生。当一个错误发生了，异常被打印出来，称为未处理异常，未处理异常会被默认处理，自动输出一些调试信息并终止运行。

(2) 检测并处理阶段。通过代码明确地处理异常，则程序不会终止运行，并能增强程序的容错性。

Python 提供 try…except 语句处理异常。Python 的 try 语句有两种：一种是处理异常(try/except/else)，另一种是无论是否发生异常都将执行最后的代码(try/finally)。

14.2.1 try…except…else 语句

try…except 语句提供了异常处理机制，保护可能导致运行时错误的某些代码行。

try…except 语法格式如下所示：

```
try:
    try 块                          #被监控的语句
except Exception[, reason]:
```

```
    except 块                #处理异常的语句
```

try 子句中的代码块放置可能出现异常的语句，except 子句中的代码块处理异常。

【例 14-4】 try…except 举例。

```
try:
    print 2/0
except ZeroDivisionError:
    print '除数不能为 0'
```

如果 try 范围内捕获了异常，就执行 except 块；如果 try 范围内没有捕获异常，就执行 else 块。

try…except…else 语法格式如下所示：

```
try:
<语句>                    #运行别的代码
except<名字>:
<语句>                    #如果引发了'name'异常，获得附加的数据
else:
<语句>                    #如果没有异常发生
```

【例 14-5】 try…except…else 举例。

```
a_list=['China', 'America', 'England', 'France']
print 'input the number of list'
while True:
    n=input()
    try:
        print a_list[n]
    except IndexError:
        print 'out of the border,please input again'
    else:
            break;
```

运行结果如图 14.1 所示。

```
C:\WINDOWS\system32\cmd.exe - PAUSE
input the number of list
8
out of  the border,please input again
5
out of  the border,please input again
3
France
请按任意键继续. . .
```

图 14.1 例 14-5 程序运行结果

多个 except 的 try 语句，语法格式如下：

```
try:
    try 块                  #被监控的语句
except Exception1:
    except 块 1             #处理异常 1 的语句
except Exception2:
    except 块 2             #处理异常 2 的语句
```

【例 14-6】 try…except…except 举例。

```
try:
    print 2/'0'
except ZeroDivisionError:
    print '除数不能为 0'
except Exception:
    print '其他类型异常'
```

为了捕获多个异常，除了声明多个 except 语句之外，还可以在一个 except 语句之后将多个异常作为元组列出来。

```
try:
    print 2/'0'
except (ZeroDivisionError,Exception):
    print '发生了一个异常'
```

14.2.2　try…finally 语句

try…finally 的用处是无论是否发生异常都要确保资源释放代码的执行。一般来说，如果没有发生错误，执行过 try 语句块之后执行 finally 语句块，完成整个流程。如果 try 语句块发生了异常，抛出了这个异常，会执行 except 语句块，然后运行 finally 语句块进行资源释放处理。

一般情况下，finally 常常用于关闭文件或者在 Socket 中。Finally 语法格式如下所示：

```
try:
    try 块                  #被监控的语句
except Exception[, reason]:
    except 块               #处理异常的语句
Finally:
            finally 组代码
```

【例 14-7】 try…except…finally 举例。

```
try:
    print 2/'0'
except (ZeroDivisionError,Exception):
    print '发生了一个异常'
```

```
finally:
    print '不管是否发生异常都执行'
```

总之，except 语句用法解释如表 14.1 所示。

表 14.1　except 语句用法

分句形式	说　　明
except	捕获所有异常类型
except name	只捕获指定类型异常
except name,value	捕获所列异常，并获得抛出的异常对象
except(name1,name2)	捕获任何列出类型的异常
except(name1,name2),value	捕获任何列出类型的异常，并获得抛出的异常对象
Else	如何没有异常发生，则运行
finally	不管有没有异常，都运行此代码块

14.3　两个特殊语句

14.3.1　raise 语句

Python 提供 raise 关键字用于引发一个异常，用法类似于 C# 和 Java 中的 throw 关键字。raise 抛出一个通用异常类型(Exception)，可以通过 dir 函数查看 exceptions 中的异常类型，如图 14.2 所示。

```
C:\WINDOWS\system32\cmd.exe - python
Microsoft Windows XP [版本 5.1.2600]
(C) 版权所有 1985-2001 Microsoft Corp.

C:\Documents and Settings\Administrator>python
Python 2.7.3 (default, Apr 10 2012, 23:31:26) [MSC v.1500 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import exceptions
>>> print dir(exceptions)
['ArithmeticError', 'AssertionError', 'AttributeError', 'BaseException', 'BufferError', 'BytesWarnin
g', 'DeprecationWarning', 'EOFError', 'EnvironmentError', 'Exception', 'FloatingPointError', 'Future
Warning', 'GeneratorExit', 'IOError', 'ImportError', 'ImportWarning', 'IndentationError', 'IndexErro
r', 'KeyError', 'KeyboardInterrupt', 'LookupError', 'MemoryError', 'NameError', 'NotImplementedError
', 'OSError', 'OverflowError', 'PendingDeprecationWarning', 'ReferenceError', 'RuntimeError', 'Runti
meWarning', 'StandardError', 'StopIteration', 'SyntaxError', 'SyntaxWarning', 'SystemError', 'System
Exit', 'TabError', 'TypeError', 'UnboundLocalError', 'UnicodeDecodeError', 'UnicodeEncodeError', 'Un
icodeError', 'UnicodeTranslateError', 'UnicodeWarning', 'UserWarning', 'ValueError', 'Warning', 'Win
dowsError', 'ZeroDivisionError', '__doc__', '__name__', '__package__']
>>>
```

图 14.2　exceptions 中的异常类型

异常类型如表 14.2 所示。

表 14.2 Python 的异常类型

异常类名	描述
NameError	引用不存在的变量
ZeroDivisionError	除数为零错误
SyntaxError	语法错误
IndexError	索引错误
KeyError	使用不存在的字典关键字
IOError	输入输出错误
ValueError	搜索列表中不存在的值
AtrributeError	调用不存在的方法
TypeError	未强制转换就混用数据类型
EOFError	文件结束标志错误

raise 语句的语法格式如下所示：

```
raise<name>                     #手工引发异常
raise<name>,<data>              #传递一个附加的数据
```

【例 14-8】 Python 提供的异常类型。

```
def factorial(n):
    if n<0:
        raise ValueError, "Expected non-negative number"
    if (n<=1):
        return 1
    else:
        return n * factorial(n-1)
```

【例 14-9】 用户自定的异常类型。

```
class ShortInputException(Exception):
    def __init__(self, length, atleast):
        Exception.__init__(self)
        self.length=length
        self.atleast=atleast
try:
    s=raw_input('please input:')
    if len(s)<3:
        raise ShortInputException(len(s), 3)
except ShortInputException, x:
     print 'ShortInputException: length is %d, At Least length is %d' % (x
     .length, x.atleast)
else:
```

```
    print 'No Exception'
```

程序运行结果如下：

```
please input:
ShortInputException: length is 0, At Least length is 3
please input:abcd
No Exception
.
```

【解析】 用户创建异常类型 ShortInputException 有 length 和 atleast 两个属性，length 是给定输入的长度，atleast 是程序期望的最小长度。当输入字符串的长度小于 3 时，由 raise 引发自定义的 ShortInputException 类异常。

14.3.2 with 语句

Python 引入 with 语句的目的是在异常处理中把 try，except 和 finally 关键字，以及与资源分配释放相关的代码全部去掉，从而减少代码的编写量，不再需要 try…except…finally 语句中的 finally 语句块，使得代码更加简洁。

with 语句的语法如下：

```
with context_expr [as var]:
    with块
```

【例 14-10】 with 举例。

```
try:
    with open('a.txt',"w") as data:
        print("File is open ")
Except IOError as err:
        print('File error:'+str(err))
```

14.4 调试

实践表明，一次性完整写完程序并运行成功的概率很小，基本不超过 1%，一般总会出现各种各样的错误需要修正。为此，需要一整套调试程序的手段来修复错误。

14.4.1 调试策略

调试过程的关键不是调试技术，而是用来推断错误原因的基本策略。调试的关键在于推断程序内部的错误位置及原因。可以采用以下方法：

(1) 试探法：针对错误列出所有可能的原因，通过测试一一排除。只要某次测试结果说明某种假设已呈现端倪，则立即精化数据，进一步进行深入的测试。

(2) 回溯法：由错误症状最先出现的地方，沿控制流往回检查，反复考虑："如果程序在这一点上的状态(变量的值)是这样，那么程序在上一点的状态一定是这样……"，直到

找到错误的位置。

（3）对分法：在关键点插入变量的正确值，根据插入位置将程序对分进行调试。

（4）归纳法调试：归纳法是一种从特殊推断一般的系统化思考方法。归纳法调试的基本思想是：从一些线索（错误征兆）着手，通过分析它们之间的关系来找出错误。

（5）强行排错：这种调试方法不需要过多的思考，目前使用较多，但效率较低。

① 在程序特定部位设置打印语句，把打印语句插在出错的源程序的各个关键变量改变部位、重要分支部位、子程序调用部位，跟踪程序的执行，监视重要变量的变化。

② 自动调试工具。利用某些程序语言的调试功能或专门的交互式调试工具，分析程序的动态过程，而不必修改程序。

14.4.2　IDLE 调试器

在 Python Shell 窗口中单击 Debug 菜单中的 Debugger 菜单项，就可以启动 IDLE 的交互式调试器，在 Debug Control 窗口中的 Python Shell 窗口中输出[DEBUG ON]并后跟一个“>>>”提示符。编程者输入的任何命令都在调试器下，在 Debug Control 窗口中查看局部变量和全局变量等有关内容。如果要退出调试器，可以再次单击 Debug 菜单中的 Debugger 菜单项，IDLE 会关闭 Debug Control 窗口，并在 Python Shell 窗口中输出[DEBUG OFF]，IDLE 调试器设置如图 14.3 所示。

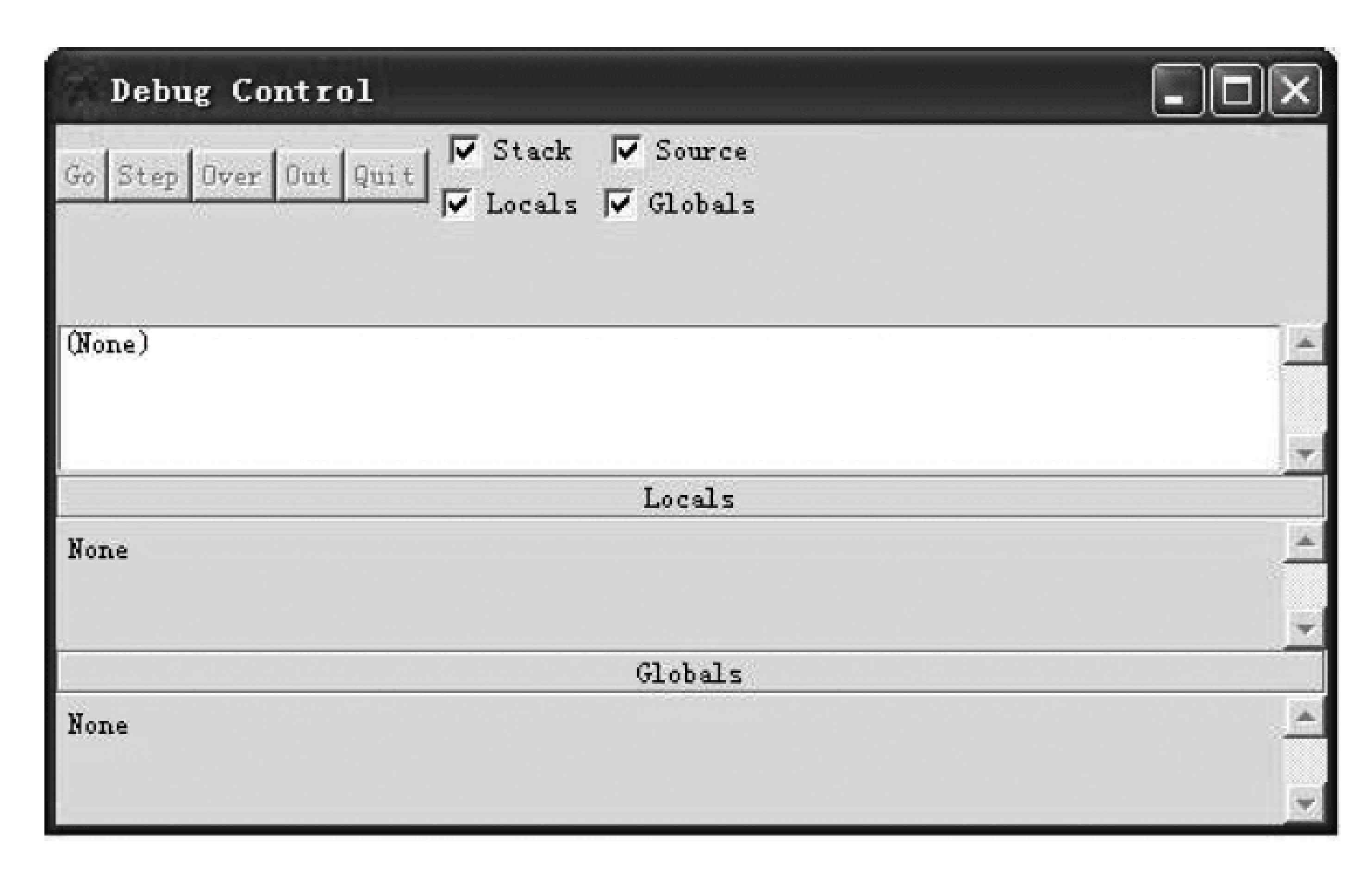


图 14.3　IDLE 调试器

14.5　习题

1. 程序设计有几种错误？分别是什么？
2. 异常处理有几种？
3. 调试策略有哪些？

参 考 文 献

[1] 赵家刚,狄光智,等.计算机编程导论：Python 程序设计[M]. 北京：人民邮电出版社,2013.

[2] 李文新,郭炜,余华山.程序设计导引及在线实践[M]. 北京：清华大学出版社,2007.

[3] 周元哲. Visual Basic.NET 程序设计[M]. 西安：西安电子科技大学出版社,2014.

[4] 周元哲. Visual Basic 程序设计语言[M]. 北京：清华大学出版社,2011.

[5] Swaroop, C. H. 著.沈洁元,译.简明 Python 教程[M]. http://sebug.net/paper/python/.

[6] (美)巴里著,林琪,等译.Head First Python(中文版)[M]. 北京：中国电力出版社,2012.

[7] 李文新,郭炜,余华山.程序设计导引及在线实践[M]. 北京：清华大学出版社,2007.

[8] 裘宗燕. 从问题到程序——程序设计与 C 语言引论[M]. 北京：机械工业出版社,2011.

[9] 王曙燕.C 语言程序设计(第二版)[M].北京：科学出版社,2008.

[10] 王红,余青松. Python 程序设计教程[M]. 北京：清华大学出版社,2014.

[11] 沙行勉.计算机科学导论——以 Python 为舟[M]. 北京：清华大学出版社,2014.

[12] (美)William F. PunchRichard Enbody 张敏,等. Python 入门经典[M]. 北京：机械工业出版社,2012.

[13] Python 中文社区[EB/OL]. http://python.cn/.

[14] Python 官方网站[EB/OL]. http://www.python.org/.

[15] Django 官方网站[EB/OL]. https://www.djangoproject.com/.

[16] 简明 Python 教程[EB/OL]. http://woodpecker.org.cn/abyteofpython_cn/chinese/.

[17] 活学活用 wxPython[EB/OL]. http://wiki.woodpecker.org.cn/moin/WxPythonInAction/.

[18] Python 编程基础[EB/OL]. http://passport.chuanke.com.

[19] WxPythonInAction[EB/OL]. http://wiki.woodpecker.org.cn/moin/WxPythonInAction.

[20] wxPython 图形用户界面开发[EB/OL]. http://www.pythoner.cn/course/topic/wxPython-gui/.

大学计算机基础教育规划教材

近　期　书　目

- 大学计算机基础(第 4 版)(“国家精品课程”、“高等教育国家级教学成果奖”配套教材、普通高等教育“十一五”国家级规划教材)
- 大学计算机基础实验指导书(“国家精品课程”、“高等教育国家级教学成果奖”配套教材)
- 大学计算机应用基础(第 2 版)(“国家精品课程”、“高等教育国家级教学成果奖”配套教材、教育部普通高等教育精品教材、普通高等教育“十一五”国家级规划教材)
- 大学计算机应用基础实验指导(“国家精品课程”、“高等教育国家级教学成果奖”配套教材)
- 大学计算机基础——计算思维初步
- 计算机程序设计基础——精讲多练 C/C++ 语言(“国家精品课程”、“高等教育国家级教学成果奖”配套教材、教育部普通高等教育精品教材、普通高等教育“十一五”国家级规划教材)
- C/C++ 语言程序设计案例教程(“国家精品课程”、“高等教育国家级教学成果奖”配套教材)
- C 程序设计(“高等教育国家级教学成果奖”配套教材、“陕西省精品课程”主讲教材、陕西普通高校优秀教材一等奖)
- C++ 程序设计(“高等教育国家级教学成果奖”配套教材)
- C# 程序设计(“高等教育国家级教学成果奖”配套教材)
- Visual Basic 2005 程序设计(“国家精品课程”、“高等教育国家级教学成果奖”配套教材、普通高等教育“十一五”国家级规划教材)
- Visual Basic 程序设计语言
- Java 语言程序设计基础(第 2 版)(普通高等教育“十一五”国家级规划教材)
- Java 语言应用开发基础(普通高等教育“十一五”国家级规划教材)
- 微机原理及接口技术(第 2 版)
- 单片机及嵌入式系统(第 2 版)
- 数据库技术及应用——Access
- SQL Server 数据库应用教程(第 2 版)(普通高等教育“十一五”国家级规划教材)
- Visual FoxPro 8.0 程序设计
- 多媒体技术及应用(“高等教育国家级教学成果奖”配套教材、普通高等教育“十一五”国家级规划教材)
- 多媒体文化基础(北京市高等教育精品教材立项项目)
- 网络应用基础(“高等教育国家级教学成果奖”配套教材)
- 计算机网络技术及应用(第 2 版)
- 计算机网络基本原理与 Internet 实践
- 可视化计算(“高等教育国家级教学成果奖”配套教材)
- Web 应用程序设计基础(第 2 版)
- Web 标准网页设计与 ASP
- MATLAB 基础教程
- Visual Basic.NET 程序设计(“高等教育国家级教学成果奖”配套教材)
- 微机原理·接口技术及应用
- Python 程序设计基础